

## A Reconstruction of Euler Data

Bong H. Lian<sup>1</sup>

*Department of Mathematics  
Brandeis University  
Waltham, MA 02154*

Chien-Hao Liu<sup>2</sup>

*Department of Mathematics  
Harvard University  
Cambridge, MA 02138*

Shing-Tung Yau<sup>3</sup>

*Department of Mathematics  
Harvard University  
Cambridge, MA 02138*

### Abstract

We apply the mirror principle of [L-L-Y] to reconstruct the Euler data  $Q = \{Q_d\}_{d \in \mathbb{N} \cup \{0\}}$  associated to a vector bundle  $V$  on  $\mathbb{CP}^n$  and a multiplicative class  $b$ . This gives a direct way to compute the intersection number  $K_d$  without referring to any other Euler data linked to  $Q$ . Here  $K_d$  is the integral of the cohomology class  $b(V_d)$  of the induced bundle  $V_d$  on a stable map moduli space. A package 'EulerData\_MP.m' in Maple V that carries out the actual computation is provided. For  $b$  the Chern polynomial, the computation of  $K_1$  for the bundle  $V = T_*\mathbb{CP}^2$ , and  $K_d$ ,  $d = 1, 2, 3$ , for the bundles  $\mathcal{O}_{\mathbb{CP}^4}(l)$  with  $6 \leq l \leq 10$  done using the code are also included.

**Key words:** Atiyah-Bott localization formula, concavex bundle, Euler data, linear  $\sigma$ -model,  $S^1 \times \mathbb{T}^n$ -equivariant cohomology.

MSC number 1991: 14Q15, 68-04, 55N91, 55R91, 81T30.

**Acknowledgements.** We would like to thank David Eisenbud, Joe Harris, Yi Hu, Sheldon Katz, Albrecht Klemm, Kefeng Liu, Jason Starr, Richard Thomas, Cumrun Vafa, and Eric Zaslow for valuable discussions at various stages of the work, Shinobu Hosono, who took the extreme pain to go through a preliminary Mathematica version of the code and gave lots of suggestions for improvement, and to Ling-Miao Chou, who went through and checked the Maple code of its near final version. Without their advices and involvements, our code may not be born yet at the moment. C.H.L. would like to thank, in addition, Hung-Wen Chang and Daniel Freed for the valuable conversation and discussion on programming before/at the beginning of the work, and Orlando Alvarez, Philip Candelas, Jacques Distler, Martin Halpern, Rafael Nepomechie, and Xenia de la Ossa, who influence and shape his view on the physics side of the subject. This work is supported by DOE grant DE-FG02-88ER25065 and NSF grants DMS-9619884 and DMS-9803347.

---

<sup>1</sup>E-mail: lian@brandeis.edu

<sup>2</sup>E-mail: chienliu@math.harvard.edu

<sup>3</sup>E-mail: yau@math.harvard.edu

## 0. Introduction and outline.

### Introduction.

Ever since the ground-breaking work of [C-dlO-G-P], mirror symmetry and its meaning and consequences have been investigated by several groups of people both from the mathematical and from the physical point of view. (See introduction of [L-L-Y] and references therein for background and for a comparison of different approaches. See also [MS].)

In this note, we apply the theory developed in the series of papers [L-L-Y] I, II, and III, to compute new intersection numbers on stable map moduli spaces. The theory goes beyond justifying mirror symmetry that relates the usually difficult A-model computations to the much more tractable B-model computations for Calabi-Yau manifolds. Indeed, the theory is a way to directly do the "A-model" computations for manifolds which are not necessarily Calabi-Yau. It is the goal of these notes to explain this and to provide a computer code that carries out the actual computations for bundles over  $\mathbb{CP}^n$ .

To make this article more self-contained, we recall in Sec. 1 the definitions of the basic objects involved and the Atiyah-Bott localization formula that is used substantially in the theory. In Sec. 2 and Sec. 3, we focus on the case of critical bundles over  $\mathbb{CP}^n$  and consider their Euler classes. In Sec. 2, we give a quick summary of facts and formula from [L-L-Y] I-III that are directly related to the actual computation of Euler data  $\{Q_d\}_d$  and the intersection numbers  $K_d$ . In Sec. 3, we explain how the theory of [L-L-Y] gives rise to a system of linear equations that can be solved inductively. The solution of the system gives the Euler data  $\{Q_d\}_d$ , from which the intersection numbers  $K_d$  can be computed. After these, we then discuss in Sec. 4 the modifications needed to take into account also non-critical bundles. There the Chern polynomial is considered. In Sec. 5, we single out six examples whose first few  $K_d$  are computed this way via a Maple code. In Sec. 6, the Maple code `EulerData_MP.m` with instructions is given. Eighteen cases have been tested and computed. The last record of the run for each of these cases is given in SEC. 3 of the code for references. The code provided can be easily modified to compute other cases of interest.

This article is served as a supplement to and a computational account of [L-L-Y]. As a result, our notations and terminologies follow [L-L-Y] very closely. Readers are referred to *ibidem* for more theoretical details.

### Outline.

1. Essential mathematical backgrounds for physicists.
2. Summary of related constructs in "Mirror Principle".
3. Computation of  $Q_d$  inductively.
4. Modifications for non-critical bundles over  $\mathbb{CP}^n$ .
5. Examples.
6. A package in Maple V for the computation of  $Q_d$  and  $K_d$ .

# 1 Essential mathematical backgrounds for physicists.

We collect in this section the most essential backgrounds for understanding these notes. Along the way, we also set up the notations for the notes.

## • Stable maps and their moduli. [Ko]

**Definition 1.1 [stable map].** Let  $X$  be a smooth projective variety. An  $n$ -pointed stable map into  $X$  consists of a connected marked curve  $(C, p_1, \dots, p_n)$  and a morphism  $f : C \rightarrow X$  satisfying the following properties:

- (1) The only singularities of  $C$  are ordinary double points.
- (2)  $p_1, \dots, p_n$  are distinct ordered smooth points of  $C$ .
- (3) If  $C_i$  is a component of  $C$  that is isomorphic  $\mathbb{CP}^1$  and is mapped to a point under  $f$ , then  $C_i$  contains at least three special (i.e. nodal or marked) points.
- (4) If  $C$  has (arithmetic) genus 1 and  $n = 0$ , then  $f$  is not constant.

*Remark 1.2.* Given Conditions (1) and (2) in the above definition, Conditions (3) and (4) are equivalent to the assertion that the data  $(f, C, p_1, \dots, p_n)$  has only finitely many automorphisms.

Given a class  $\beta \in H_2(X, \mathbb{Z})$ , the moduli space of all stable maps  $(f, C, p_1, \dots, p_n)$  such that  $[f(C)] = \beta$  into  $X$  will be denoted by  $\overline{\mathcal{M}}_{g,n}(X, \beta)$ .

• **Equivariant cohomology.** (See [Au].) Given a group  $G$  acting on a space  $X$ . Let  $BG$  be the classifying space and  $EG \rightarrow BG$  be the universal principle  $G$ -bundle associated to  $G$ . The equivariant cohomology  $H_G^*(X)$  of  $X$  associated to the  $G$ -action is defined to be

$$H_G^*(X) = H^*(X_G),$$

where  $X_G = EG \times_G X$  is the total space of the associated  $X$ -bundle over  $BG$ . Note that  $H_G^*(pt) = H^*(BG)$ , and that  $H_G^*(X)$  is naturally a  $H_G^*(pt)$ -module.

The constant map  $X \rightarrow pt$  induces an equivariant projection  $\pi_X : X_G \rightarrow BG$ . The induced pushforward map  $\pi_{X!}$  from  $H_G^*(X)$  to  $H_G^*(pt)$  is given by integration along the fiber of  $\pi_X$ . This is also called the *equivariant integral*. In notation,

$$\pi_{X!} = \int_{X_G} : H_G^*(X) \longrightarrow H_G^*(pt).$$

*Remark 1.3.* In this article, the coefficient for  $H^*(X_G)$  can be  $\mathbb{Q}$ ,  $\mathbb{R}$ , or  $\mathbb{C}$ . Usually we use  $\mathbb{Q}$  or  $\mathbb{C}$  in the discussion.

**Example 1.4.** Let  $\mathbb{T}^r = \prod_r S^1$  be an  $r$ -torus. Then  $B\mathbb{T}^r = \prod_r \mathbb{CP}^\infty$  and  $H^*(B\mathbb{T}^r) = H_{\mathbb{T}^r}^*(pt) = \mathbb{C}[\lambda_1, \dots, \lambda_r]$ , the polynomial ring generated by  $\lambda_1, \dots, \lambda_r$ , where  $\lambda_i$  is the first Chern class of the hyperplane line bundle  $\mathcal{O}(1)$  over the  $i^{th}$   $\mathbb{CP}^\infty$  in the product.

Let  $\mathbb{T}^r \rightarrow GL(N+1, \mathbb{C})$  be a representation of an  $r$ -torus on  $\mathbb{C}^{N+1}$  with weight  $(\beta_0, \dots, \beta_N)$ . Note that each  $\beta_i$  is a linear combination of the  $\lambda_j$ 's. This induces a  $\mathbb{T}^r$ -action on  $\mathbb{CP}^N$ . With respect to this action,

$$H_{\mathbb{T}^r}^*(\mathbb{CP}^N) = H_{\mathbb{T}^r}^*(pt)[\zeta] / \prod_{i=0}^N (\zeta - \beta_i),$$

where  $\zeta$  is the equivariant hyperplane class from a fixed lifting of the hyperplane class of  $\mathbb{CP}^N$ . The equivariant integral  $\int_{(\mathbb{CP}^N)_{\mathbb{T}^r}} : H_{\mathbb{T}^r}^*(\mathbb{CP}^N) \rightarrow H_{\mathbb{T}^r}^*(pt)$  picks out the coefficient of  $\zeta^N$  in elements of  $H_{\mathbb{T}^r}^*(\mathbb{CP}^N)$ . □

• **The Atiyah-Bott localization formula.** [A-B] Let  $T$  be an  $r$ -torus that acts on a manifold  $X$  with the set of fixed points a union of smooth connected submanifolds  $Z_j$ . Then the normal bundle  $N_j$  of  $Z_j$  in  $X$  is a  $T$ -equivariant vector bundle with its equivariant Euler class  $e_T(N_j) \in H_T^*(Z_j)$ .

There are three fundamental maps between  $H_T^*(Z_j)$  and  $H_T^*(X)$ :

(1) *the restriction homomorphism:*

$$i_j^* : H_T^*(X) \longrightarrow H_T^*(Z_j)$$

induced by the equivariant inclusion  $i_j : Z_j \hookrightarrow X$ ;

(2) *the Gysin map:*

$$i_{j!} : H_T^*(Z_j) \longrightarrow H_T^*(X);$$

Note that ([Au]), for any  $\alpha_j \in H_T^*(Z_j)$ , one has

$$i_j^* \circ i_{j!}(\alpha) = \alpha_j \cup e_T(N_j).$$

Let  $\mathcal{R}$  be the localization of  $H_T^*(BT)$ . With the notation following Example 1.4,  $\mathcal{R} = \mathbb{C}(\lambda_1, \dots, \lambda_r)$ . Then  $e_T(N_j)$  is an invertible element in the localization  $H_T^*(Z_j) \otimes \mathcal{R}$ . We can now state the Atiyah-Bott localization formula [A-B]:

**Fact 1.5 [Atiyah-Bott localization formula].** *The following map is an isomorphism*

$$\begin{aligned} H_T^*(X) \otimes \mathcal{R} &\xrightarrow{\sim} \bigoplus_j H_T^*(Z_j) \otimes \mathcal{R} \\ \alpha &\longmapsto \left( \frac{i_j^*(\alpha)}{e_T(N_j)} \right)_j. \end{aligned}$$

*Its inverse is given by*

$$(\alpha_j)_j \longmapsto \sum_j i_{j!}(\alpha_j).$$

*Combining these two, one has*

$$\alpha = \sum_j i_{j!} \left( \frac{i_j^*(\alpha)}{e_T(N_j)} \right),$$

for any  $\alpha \in H_T^*(X) \otimes \mathcal{R}$ .

**Example 1.6.** (*Continuing Example 1.4*). The fixed point set on  $\mathbb{CP}^N$  comes from the  $N+1$  coordinate lines of  $\mathbb{C}^{N+1}$ . Denote this set by  $\{p_0, \dots, p_N\}$ . Let  $\alpha \in H_T^*(\mathbb{CP}^N) \otimes \mathcal{R}$ , then  $\alpha$  can be written as a polynomial  $f(\zeta)$  with coefficients in  $\mathcal{R}$ . In terms of this,  $i_j^*(\alpha) = f(\beta_j)$ ;  $e_T(N_j) = \prod_{k \neq j} (\beta_j - \beta_k)$ , where  $k$  runs in  $\{0, \dots, N\}$ ; and the Gysin map  $i_{j!}$  is given by the cup product with  $\prod_{k \neq j} (\zeta - \beta_k)$ . Thus, from the localization formula, one has

$$f(\zeta) = \sum_{j=0}^N f(\beta_j) \frac{\prod_{k \neq j} (\zeta - \beta_k)}{\prod_{k \neq j} (\beta_j - \beta_k)}.$$

□

• **Concavex bundles over  $\mathbb{CP}^n$ .** ([L-L-Y, I].) Let  $T = \mathbb{T}^{n+1}$  acts on  $\mathbb{C}^{n+1}$  with weights  $\lambda_0, \dots, \lambda_n$ . It induces an action on  $\mathbb{CP}^n$ .

**Definition 1.7 [concavex bundle].** Let  $V$  be a  $T$ -equivariant vector bundle over  $\mathbb{CP}^n$ . We call  $V$  *convex* (resp. *concave*) if the  $T$ -equivariant Euler class  $e_T(V)$  is invertible and  $H^1(C, f^*V) = 0$  (resp.  $H^0(C, f^*V) = 0$ ) for any 0-pointed genus 0 stable map  $f : C \rightarrow \mathbb{CP}^n$ . We call  $V$  *concavex* if it is a direct sum of a convex and a concave bundle. We denote this decomposition by  $V = V^+ \oplus V^-$ .

**Definition 1.8 [splitting type].** Let  $V$  be a  $T$ -equivariant concavex bundle over  $\mathbb{CP}^n$ . Let  $l_a, k_b$  be positive integers such that for every  $T$ -invariant line  $C \cong \mathbb{CP}^1$  in  $\mathbb{CP}^n$  we have a  $T$ -equivariant isomorphism

$$V|_C \cong \oplus_a \mathcal{O}(l_a) \oplus \oplus_b \mathcal{O}(-k_b).$$

Then we call  $(l_1, \dots; k_1, \dots)$  the *splitting type* of  $V$ .

## 2 Summary of related constructs in “Mirror Principle”.

This section follows [L-L-Y, I]. Readers should consult [L-L-Y, I] (see also [L-L-Y, II]) for more details.

• **Set-up of the problem.** Let  $T = \mathbb{T}^{n+1}$ ,  $V$  be a concavex  $T$ -equivariant bundle over  $\mathbb{CP}^n$  of splitting type  $(l_1, \dots; k_1, \dots)$ , and  $d$  be a class in  $H_2(\mathbb{CP}^n, \mathbb{Z}) = \mathbb{Z}$ . Let  $\overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d)$  be the moduli space of stable maps into  $\mathbb{CP}^n$ , of degree  $d$ , genus 0, without marked points; and similarly for  $\overline{\mathcal{M}}_{0,1}(\mathbb{CP}^n, d)$  and  $M_d = \overline{\mathcal{M}}_{0,0}(\mathbb{CP}^1 \times \mathbb{CP}^n, (1, d))$ . Let  $N_d = \mathbb{CP}^{(n+1)d+n}$  be the linear  $\sigma$ -model for  $\mathbb{CP}^n$ . (We will say more about  $N_d$  in the next

item.) Then one has the following  $S^1 \times T$ -equivariant diagram:

$$\begin{array}{ccccccc}
& V_d = \pi^* U_d & & U_d & & \rho^* U_d & & V \\
& \downarrow & & \downarrow & & \downarrow & & \downarrow \\
N_d & \xleftarrow{\varphi} & M_d & \xrightarrow{\pi} & \overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d) & \xleftarrow{\rho} & \overline{\mathcal{M}}_{0,1}(\mathbb{CP}^n, d) & \xrightarrow{ev} & \mathbb{CP}^n \\
& & \parallel & & & & & & \\
& & \overline{\mathcal{M}}_{0,0}(\mathbb{CP}^1 \times \mathbb{CP}^n, (1, d)) & & & & & & ,
\end{array}$$

where  $\rho$  forgets and  $ev$  evaluates at the marked point of a 1-pointed stable map;  $U_d = \rho_! ev^* V$ , the pushforward via  $\rho$  of the pullback of  $V$  via  $ev$ ;  $\pi$ , the *contracting morphism*, is induced by the projection of a stable map in  $\mathbb{CP}^1 \times \mathbb{CP}^n$  to the  $\mathbb{CP}^n$  component and contracting the unstable components; and  $\varphi$ , the *collapsing morphism*, will be explained in the next two items.

Let  $c_{top}$  be the top Chern class (i.e. the Euler class) of  $U_d$ , then the *intersection number of degree  $d$*  is defined to be

$$K_d = \int_{\overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d)} c_{top}(U_d).$$

One of the goals in the mirror symmetry literatures is to compute  $K_d$ 's and to relate them to enumerative problems on  $\mathbb{CP}^n$ , or some varieties therein. An important insight from [L-L-Y] is that one can reduce this problem to an easy problem on projective spaces  $\{N_d\}_{d=0}^\infty$ , called the linear  $\sigma$ -model in [L-L-Y]. In fact the intersection numbers  $K_d$  can be recovered from cohomology classes  $Q = \{Q_d\}_{d=0}^\infty$ , called Euler data, defined on those projective spaces. In turn the Euler data can be computed essentially by an elementary algorithm, and, sometimes, via an explicit formula. A nonlinear recursion involving graph sums was used to compute  $K_4$  in the case of  $O(5)$  on  $\mathbb{CP}^4$  in [Ko].

If  $b$  is any multiplicative cohomology classes, then we can apply our algorithm to compute the integrals

$$K_d = \int_{\overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d)} b(U_d).$$

More generally, suppose  $a, b, \dots$  are any multiplicative cohomology classes. Then, for any given vector bundle  $V$ , we have

$$a(V) = a_0(V) + a_1(V) + \dots + a_r(V),$$

where  $a_i(V)$  is the degree  $i$  component of  $a(V)$ . We can homogenize  $a$  by writing

$$a_x(V) = \sum_i x^{r-i} a_i(V),$$

where  $x$  is a formal variable, and the class  $a_x$  remains multiplicative. Likewise, we have

$$b_y(V) = \sum_i y^{r-i} b_i(V),$$

etc. Multiplying them, we get  $a_x(V)b_y(V)\cdots$ , which is a new multiplicative class. Note that this is a polynomial in the variables  $x, y, \dots$ , with coefficients of the form  $a_i(V)b_j(V)\cdots$ . Moreover, each such product correspond to a unique monomial in  $x, y, \dots$ . Now our algorithm computes  $K_d \in \mathbb{C}[x, y, \dots]$  for the multiplicative class  $a_x(U_d)b_y(U_d)\cdots$  and, hence, computes all coefficients of the form

$$\int_{\overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d)} a_i(U_d)b_j(U_d), \dots$$

Note that this number is zero unless the integrand has the right total degree.

*Notation.* (Cf. Example 1.4.) We shall adapt the following notations for the rest of the article:  $G = S^1 \times T$ ,  $\lambda = (\lambda_0, \dots, \lambda_n)$ ,  $\alpha = c_1(\mathcal{O}(1)) \in H_{S^1}^*(BS^1)$ ,  $\mathcal{R} = \mathbb{Q}(\lambda)[\alpha]$ ,  $\mathcal{R}^{-1} = \mathbb{Q}(\lambda, \alpha)$ ,  $\mathcal{R}H_G^*(\cdot) = H_G^*(\cdot) \otimes_{\mathbb{Q}[\lambda, \alpha]} \mathcal{R}$ ,  $\mathcal{R}^{-1}H_G^*(\cdot) = H_G^*(\cdot) \otimes_{\mathbb{Q}[\lambda, \alpha]} \mathcal{R}^{-1}$ .

• **The linear  $\sigma$ -model  $\{N_d\}_{d=0}^\infty$  for  $\mathbb{CP}^n$ .** Let

$$N_d = \mathbb{P}(H^0(\mathbb{CP}^1, \mathcal{O}_{\mathbb{CP}^1}(d))^{n+1}) \cong \mathbb{CP}^{(n+1)d+n}$$

be the space of  $(n+1)$ -tuple of homogeneous polynomials of degree  $d$  on  $\mathbb{CP}^1$  up to an overall constant multiple in  $\mathbb{C}$ . An element in  $N_d$  can be written as

$$[\sum_r z_{0r}w_0^r w_1^{d-r} : \cdots : \sum_r z_{nr}w_0^r w_1^{d-r}],$$

where  $[w_0 : w_1]$  is the homogeneous coordinates for  $\mathbb{CP}^1$  and  $z_{ir} \in \mathbb{C}$ . The sequence  $\{N_d\}_{d=0}^\infty$  is called *the linear  $\sigma$ -model* for  $\mathbb{CP}^n$ .

Let  $G = S^1 \times \mathbb{T}^{n+1}$ ; then  $G$  acts on  $\mathbb{CP}^1 \times \mathbb{CP}^n$  by

$$(t, t_0, \dots, t_n) \cdot ([w_0 : w_1], [x_0 : \cdots : x_n]) = ([tw_0 : w_1], [t_0x_0 : \cdots : t_nx_n]).$$

This induces a  $G$ -action on  $N_d$  with fixed points

$$p_{i,r} = [0 : \cdots : 0 : w_0^r w_1^{d-r} : 0 : \cdots : 0],$$

where the non-zero term appears at the  $i^{th}$  position,  $i = 0, \dots, n$ , and  $r = 0, \dots, d$ . Note that the weight for the  $G$ -action at  $T_{p_{i,r}}N_d$  is  $\lambda_i + r\alpha$ .

There are two  $G$ -equivariant maps between the  $N_d$ 's, defined as follows:

$$\begin{aligned} I : N_{d-1} &\rightarrow N_d, & [f_0 : \cdots : f_n] &\mapsto [w_1 f_0 : \cdots : w_1 f_n]; & \text{and} \\ \neg : N_d &\rightarrow N_d, & [f_0(w_0, w_1) : \cdots : f_n(w_0, w_1)] &\mapsto [f_0(w_1, w_0) : \cdots : f_n(w_1, w_0)]. \end{aligned}$$

From  $I$ , one obtains a chain of inclusions

$$N_0 = \mathbb{CP}^n \xrightarrow{I} N_1 \xrightarrow{I} \cdots \xrightarrow{I} N_d,$$

whose composition gives a canonical inclusion  $I_d : N_0 = \mathbb{CP}^n \rightarrow N_d$ . Let  $\kappa$  be the equivariant hyperplane class in  $H_G^*(N_d)$ . Then the induced map of  $\neg$  on  $\mathcal{R}^{-1}H_G^*(N_d)$  is generated by  $\overline{\kappa} = \kappa - d\alpha$ ,  $\overline{\alpha} = -\alpha$ , and  $\overline{\lambda}_i = \lambda_i$ .

• **The  $G$ -equivariant morphism  $\varphi : M_d \rightarrow N_d$ .** First note that  $M_d$  and  $N_d$  are two different compactifications of the space  $\mathcal{M}_{0,0}(\mathbb{CP}^n, d)$  of degree  $d$  maps from  $\mathbb{CP}^1$  to  $\mathbb{CP}^n$ . Precisely, an element in  $\mathcal{M}_{0,0}(\mathbb{CP}^n, d)$  can be written as

$$[f_0 : \cdots : f_n] = [\sum_r z_{0r} w_0^r w_1^{d-r} : \cdots : \sum_r z_{nr} w_0^r w_1^{d-r}]$$

with  $f_0, \dots, f_n$  relatively prime. Its embedding in  $N_d$  is tautological, while its embedding in  $M_d$  is given by

$$[f_0 : \cdots : f_n] \mapsto ([w_1 : w_0], [f_0 : \cdots : f_n]).$$

Via these embeddings, the identity map on  $\mathcal{M}_{0,0}(\mathbb{CP}^n, d)$  extends to a  $G$ -equivariant morphism  $\varphi : M_d \rightarrow N_d$ . Explicitly,  $\varphi$  can be described as follows:

Let  $(f, C) \in M_d$  and  $\pi_1, \pi_2$  be the projections of  $\mathbb{CP}^1 \times \mathbb{CP}^n$  onto its first and second factor respectively. Then one can decompose  $C$  into  $C_0 \cup C_1 \cup \cdots \cup C_s$  with  $C_0 \cap C_j = x_j$  for  $j > 0$  such that  $\pi_1 \circ f : C_0 \xrightarrow{\sim} \mathbb{CP}^1$  and any other  $C_j$  is pinched to some  $\pi_1 \circ f(x_j) = [a_j, b_j] \in \mathbb{CP}^1$  under  $\pi_1 \circ f$ . Let  $d_i$  be the degree of  $\pi_2 \circ f : C_j \rightarrow \mathbb{CP}^n$  and  $[\sigma_0 : \cdots : \sigma_n]$  represent the degree  $d_0$  map  $\pi_2 \circ f : C_0 \rightarrow \mathbb{CP}^n$ . Then

$$\varphi : (f, C) \mapsto [g \sigma_0 : \cdots : g \sigma_n], \quad \text{where} \quad g = \prod_{j=1}^s (a_j w_0 - b_j w_1)^{d_j}.$$

• **Euler data.**

**Definition 2.1 [Euler data].** Given an invertible class  $\Omega \in H_T^*(\mathbb{CP}^n)^{-1}$ , the localization of  $H_T^*(\mathbb{CP}^n)$ , an  $\Omega$ -Euler data is a sequence  $Q = \{Q_d\}_{d=0}^\infty$  of classes  $Q_d \in \mathcal{RH}_G^*(N_d)$  that satisfy

- (1)  $Q_0 = \Omega$ .
- (2) The *gluing identity*:

$$i_{p_i}^*(\Omega) i_{p_{i,r}}^*(Q_d) = \overline{i_{p_{i,0}}^*(Q_r)} i_{p_{i,0}}^*(Q_{d-r}),$$

for all  $d$  and  $i = 0, \dots, n, r = 0 \dots d$ .

An immediate consequence is the following lemma:

**Fact 2.2 [reciprocity].** (Lemma 2.4 in [L-L-Y, I].) *If  $Q$  is an Euler data, then, for  $i, j = 0, \dots, n, r = 0, \dots, d, d = 0, 1, 2, \dots$ , one has*

- (1)  $Q_d(\lambda_i + d\alpha) = \overline{Q_d(\lambda_i)}$ .
- (2)  $Q_d(\lambda_i)|_{\alpha=(\lambda_i-\lambda_j)/d} = Q_d(\lambda_j)|_{\alpha=(\lambda_j-\lambda_i)/d}$  for  $d > 0$ .
- (3)  $\Omega(\lambda_i) Q_d(\lambda_j) = Q_r(\lambda_j) Q_{d-r}(\lambda_i)$  at  $\alpha = (\lambda_j - \lambda_i)/r$  for  $r > 0$ .



Recall the various bundles and maps from Item Set-up.

**Fact 2.3 [Euler data].** (Theorem 2.8 in [L-L-Y, I].) *Let  $V = V^+ \oplus V^-$  be a concave bundle over  $\mathbb{CP}^n$ ,  $\chi_d^V$  be the equivariant Euler class of  $V_d$ ,  $Q_0 = \Omega^V = e_T(V^+)/e_T(V^-)$ ,  $Q_d = \varphi!(\chi_d^V)$  for  $d > 0$ . Then  $Q = \{Q_d\}$  is an  $\Omega^V$ -Euler data.*

We call a concave bundle  $V \rightarrow \mathbb{CP}^n$  *critical* if the induced bundle  $U_d \rightarrow \overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d)$  has rank equal to  $\dim(\overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d)) = (n+1)d + n - 3$ .

**Fact 2.4.** (Theorem 3.2 (ii) in [L-L-Y, I].) *Let  $V$  be a critical concave bundle over  $\mathbb{CP}^n$ . Then in the non-equivariant limit  $\lambda \rightarrow 0$ ,*

$$\int_{\mathbb{CP}^n} e^{-Ht/\alpha} \frac{\lim_{\lambda \rightarrow 0} I_d^*(Q_d)}{\prod_{m=1}^d (H - m\alpha)^{n+1}} = \alpha^{-3} (2 - dt) K_d.$$

Thus, once  $Q_d$  is determined, the intersection number  $K_d$  is also determined.

*Remark 2.5.* Since  $H^{n+1} = 0$ , one can rewrite the above formula as

$$\int_{\mathbb{CP}^n} \left[ \sum_{k=0}^n \frac{(-Ht/\alpha)^k}{k!} \right] \left[ \lim_{\lambda \rightarrow 0} I_d^*(Q_d) \right] \frac{(-1)^{(n+1)d}}{(d!)^{n+1} \alpha^{(n+1)d}} \left[ \sum_{k=0}^n \left( \frac{H}{m\alpha} \right)^k \right]^{n+1} = \alpha^{-3} (2 - dt) K_d.$$

Note also that in [C-K], there is another formula, implicitly in [L-L-Y, I], that relates  $Q_d$  and  $K_d$ :

$$\int_{\mathbb{CP}^n} H e^{-Ht/\alpha} \frac{\lim_{\lambda \rightarrow 0} I_d^*(Q_d)}{\prod_{m=1}^d (H - m\alpha)^{n+1}} = \alpha^{-2} d K_d.$$

• **Determination of an Euler data.** By the localization formula,  $Q_d$  is determined by its restriction  $i_{p_{i,r}}^*(Q_d)$  at the fixed points  $p_{i,r}$ , for  $i = 0, \dots, n$ ,  $r = 0, \dots, d$ . Explicitly,

$$Q_d = \sum_{(i,r)} \frac{i_{p_{i,r}}^*(Q_d) \prod_{(j,s) \neq (i,r)} (\kappa - \lambda_j - s\alpha)}{\prod_{(j,s) \neq (i,r)} (\lambda_i - \lambda_j + (r-s)\alpha)}.$$

Since the Euler data condition says that

$$i_{p_{i,r}}^*(Q_d) = \frac{\overline{i_{p_{i,0}}^*(Q_r)} i_{p_{i,0}}^*(Q_{d-r})}{i_{p_i}^*(\Omega)},$$

it turns out that, to determine  $Q_d$ , one only needs to know its restrictions  $i_{p_{i,0}}^*(Q_d)$  at  $p_{i,0}$  for  $i = 0, \dots, n$ .

We can now state theorems from [L-L-Y, I] that enables one to determine  $i_{p_{i,0}}^*(Q_d)$ .

**Fact 2.6 [degree bound and determination of Euler data].** (Theorem 2.10, Theorem 2.11, and Theorem 3.2 (i) in [L-L-Y, I].) *Let  $V$  be a concave bundle over  $\mathbb{CP}^n$  of splitting*

type  $(l_1, l_2, \dots; k_1, k_2, \dots)$ ,  $\chi$  be its Euler characteristic class, and  $Q = \{Q_d\}_{d \in \mathbb{N} \cup \{0\}}$  be the  $\chi$ -Euler data for  $V$ , as in Fact 2.3. Then the restrictions  $I_d^*(Q_d) \in H_G^*(\mathbb{CP}^n)$  has

$$\deg_\alpha I_d^*(Q_d) \leq (n+1)d - 2.$$

Furthermore,  $Q$  is completely determined by the value of the restrictions  $i_{p_{i,0}}^*(Q_d)$ ,  $i = 0, \dots, n$ ,  $d = 0, 1, 2, \dots$ , at  $\alpha = (\lambda_i - \lambda_j)/d$ ,  $i \neq j$ . These values are given explicitly by

$$i_{p_{i,0}}^*(Q_d) \Big|_{\alpha = \frac{\lambda_i - \lambda_j}{d}} = \prod_a \prod_{m=0}^{l_a d} (l_a \lambda_i - m \frac{\lambda_i - \lambda_j}{d}) \prod_b \prod_{m=1}^{k_b d - 1} (-k_b \lambda_i + m \frac{\lambda_i - \lambda_j}{d}).$$

*Remark 2.7 [total degree bound].* For  $V$  critical, since the rank of  $U_d \rightarrow \overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d)$  is  $(n+1)d + n - 3$ , the total degree of  $Q_d$ , as a polynomial of  $\kappa$ ,  $\alpha$ , and  $\lambda$ , is bounded by  $(n+1)d + n - 3$ .

These facts and remarks allow one to compute  $Q_d$  as a polynomial of  $\kappa$  and  $\alpha$  with coefficients in  $\mathbb{C}(\lambda_0, \dots, \lambda_n)$ . We now turn to this detail.

### 3 Computation of $Q_d$ inductively.

Following previous notations, let  $V$  be a critical concavex bundle over  $\mathbb{CP}^n$  of splitting type  $(l_1, l_2, \dots; k_1, k_2, \dots)$ . Then  $Q_1$  can be computed, using the Atiyah-Bott formula. The higher  $Q_d$  can be computed by the recursive relation from the gluing identities, the special values of  $Q_d$  at the fixed points, and the  $\alpha$  degree bound of  $I_d^*(Q_d)$ .

#### The computation of $Q_1$ .

For  $d = 1$ ,  $\deg_\alpha Q_1(\lambda_i, \alpha) \leq n - 1$  and, for a fixed  $i$ , the  $n$ -many values  $Q_1(\lambda_i, \lambda_i - \lambda_j)$ ,  $j \neq i$  are known from Fact 2.6:

$$Q_1(\lambda_i, \lambda_i - \lambda_j) = \prod_a \prod_{m=0}^{l_a} (l_a \lambda_i - m (\lambda_i - \lambda_j)) \prod_b \prod_{m=1}^{k_b - 1} (-k_b \lambda_i + m (\lambda_i - \lambda_j)), \quad \text{for } j \neq i.$$

Thus, using the Lagrange interpolation formula, one obtains

$$i_{p_{i,0}}^*(Q_1) = Q_1(\lambda_i, \alpha) = \sum_{\substack{j=0, \dots, n \\ j \neq i}} Q_1(\lambda_i, \lambda_i - \lambda_j) \frac{\prod_{k \neq i, j} (\alpha - \lambda_i + \lambda_k)}{\prod_{k \neq i, j} (\lambda_k - \lambda_j)}.$$

By the Reciprocity Lemma,  $i_{p_{i,1}}^*(Q_1) = \overline{Q_1(\lambda_i, \alpha)} = Q_1(\lambda_i, -\alpha)$ . In this way, the restriction of  $Q_1$  at the set of fixed points of the  $S^1 \times \mathbb{T}^{n+1}$ -action on  $N_1$  are all acquired.

Using the localization formula and playing around with the indices, one obtains an exact expression

$$Q_1 = \sum_{i=0}^n \left[ (f_i(\alpha)(\kappa - \lambda_i - \alpha) + f_i(-\alpha)(\kappa - \lambda_i)) \prod_{j \neq i} (\kappa - \lambda_j) \prod_{j \neq i} (\kappa - \lambda_j - \alpha) \right],$$

where

$$\begin{aligned} f_i(\alpha) &= \frac{i_{p_{i,0}}^*(Q_1)}{\prod_{(j,s) \neq (i,0)} (\lambda_i - \lambda_j - s\alpha)} \\ &= \sum_{j \neq i} \frac{\prod_a \prod_{m=0}^{l_a} (l_a \lambda_i - m(\lambda_i - \lambda_j)) \prod_b \prod_{m=1}^{k_b-1} (-k_b \lambda_i + m(\lambda_i - \lambda_j))}{\alpha(\alpha - \lambda_i + \lambda_j) \prod_{k \neq i} (\lambda_i - \lambda_k) \prod_{k \neq i, j} (\lambda_j - \lambda_k)}. \end{aligned}$$

### The computation of $Q_d$ for $d > 1$ .

Let  $N = (n+1)d + n - 3$ . Then one may write  $Q_d$  as a polynomial in  $\kappa, \alpha$  with coefficients in  $\mathbb{C}[\lambda]$ :

$$Q_d = \sum_{\mu=0}^N \sum_{\nu=0}^{N-\mu} w_{\mu\nu} \alpha^\mu \kappa^\nu \quad \text{with} \quad w_{\mu\nu} \in \mathbb{C}[\lambda].$$

Since we have an explicit formula for  $Q_1$ , we may assume that  $Q_0, Q_1, \dots, Q_{d-1}$  are all determined.

• *Systems from the gluing identity*: The gluing identity that an Euler data must satisfy is completely encoded in the following two systems of linear equations in  $w_{\mu\nu}$ :

- (1)  $Q_d(\lambda_i + r\alpha, \alpha)$  for  $i = 0, \dots, n$  and  $r = 1, \dots, d-1$ :

The gluing identity says

$$Q_d(\lambda_i + r\alpha, \alpha) = \sum_{\mu=0}^N \sum_{\nu=0}^{N-\mu} w_{\mu\nu} \alpha^\mu (\lambda_i + r\alpha)^\nu = i_{p_{i,r}}(Q_d) = \overline{Q_r(\lambda_i, \alpha)} Q_{d-r}(\lambda_i, \alpha) / \Omega(\lambda_i)$$

for  $i = 0, \dots, n$  and  $r = 1, \dots, d-1$ . Denote  $\overline{Q_r(\lambda_i, \alpha)} Q_{d-r}(\lambda_i, \alpha) / \Omega(\lambda_i)$  by  $b_1(i, r)$ ; then  $b_1(i, r)$  is known by induction, Furthermore,

$$\deg_\alpha(b_1(i, r)) \leq (n+1)r - 2 + (n+1)(d-r) - 2 = (n+1)d - 4 < N.$$

Thus, one may write  $b_1(i, r) = \sum_{s=0}^N b_1(i, r, s) \alpha^s$ , where  $b_1(i, r, s) = 0$  for  $s \geq (n+1)d - 3$ . After expanding the powers and exchanging and relabelling the indices to the above equation, one obtains the following linear system in  $w_{\mu\nu}$ :

$$\left\{ \begin{array}{l} \sum_{\mu=0}^s \sum_{\nu=s-\mu}^{N-\mu} w_{\mu\nu} \cdot \binom{\nu}{s-\mu} r^{s-\mu} \lambda_i^{\mu+\nu-s} = b_1(i, r, s) \\ \text{for } i = 0, \dots, n, r = 1, \dots, d-1, \text{ and } s = 0, \dots, N. \end{array} \right.$$

(2)  $Q_d(\lambda_i + d\alpha, \alpha)$  for  $i = 0, \dots, n$ :

The gluing identity says

$$Q_d(\lambda_i + d\alpha, \alpha) = \sum_{\mu=0}^N \sum_{\nu=0}^{N-\mu} w_{\mu\nu} \alpha^\mu (\lambda_i + d\alpha)^\nu = \overline{Q_d(\lambda_i, \alpha)} = Q_d(\lambda_i, -\alpha) = \sum_{\mu=0}^N \sum_{\nu=0}^{N-\mu} w_{\mu\nu} (-\alpha)^\mu \lambda_i^\nu$$

for  $i = 0, \dots, n$ . This gives rise to the second linear system in  $w_{\mu\nu}$ :

$$\left\{ \begin{array}{l} \sum_{\mu=0}^{s-1} \sum_{\nu=s-\mu}^{N-\mu} w_{\mu\nu} \cdot \binom{\nu}{s-\mu} d^{s-\mu} \lambda_i^{\mu+\nu-s} + \sum_{\nu=0}^{N-s} w_{s\nu} \cdot \left(1 + (-1)^{s+1}\right) \lambda_i^\nu = 0 \\ \text{for } i = 0, \dots, n \text{ and } s = 0, \dots, N. \end{array} \right.$$

• *System from the special values:*

(3)  $Q_d(\lambda_i, \frac{\lambda_i - \lambda_j}{d})$  for  $i = 0, \dots, n$ :

Fact 2.6 gives rise to the third linear system in  $w_{\mu\nu}$ :

$$\left\{ \begin{array}{l} \sum_{\mu=0}^N \sum_{\nu=0}^{N-\mu} w_{\mu\nu} \cdot \left(\frac{\lambda_i - \lambda_j}{d}\right)^\mu \lambda_i^\nu = \prod_a \prod_{m=0}^{l_a d} (l_a \lambda_i - m \frac{\lambda_i - \lambda_j}{d}) \\ \text{for } i, j = 0, \dots, n, i \neq j. \end{array} \right.$$

• *System from the  $\alpha$ -degree bound:*

(4)  $\deg_\alpha I_d^*(Q_d) \leq (n+1)d - 2$  :

Since  $H_G^*(\mathbb{CP}^n) = \mathbb{C}[\lambda, \alpha][\kappa] / \prod_{i=0}^n (\kappa - \lambda_i)$ ,  $I_d^*(Q_d)$  is obtained by  $Q_d$  modulo the relation  $\prod_{i=0}^n (\kappa - \lambda_i) = 0$ . This is achieved by iterations of the set of replacements

$$\{ \kappa^{n+1+i} \rightarrow \kappa^i (-\kappa^{n+1} + \prod_{i=0}^n (\kappa - \lambda_i)) \mid i = 0, \dots, N - n - 1 \}$$

until the  $\kappa$ -degree of the resulting  $Q_d$  is less than  $n + 1$ . This can be easily done by computer. In this way, one obtains  $I_d^*(Q_d)$ . Let

$$I_d^*(Q_d) = \sum_{i,j} w'_{ij} \alpha^i \kappa^j.$$

Then  $w'_{ij}$  is a linear combination of  $w_{\mu\nu}$  with coefficients in  $\mathbb{C}[\lambda]$ . The  $\alpha$ -degree bound then gives us the fourth system of linear equations in  $w_{\mu\nu}$ :

$$\left\{ \begin{array}{ll} w'_{ij} = 0, & \text{for } i \geq (n+1)d-1. \end{array} \right.$$

*Remark 3.1.* Note that the whole content of gluing identities in the definition of Euler data is already absorbed in the first and the second linear systems (1) and (2) above. Since all the identities that appear in the Reciprocity Lemma are obtained by substituting into the gluing identities some special  $\alpha$  values, they will be automatically satisfied once System (1) and System (2) above are satisfied. Thus, they do not provide us with extra equations. Furthermore, Fact 2.6 implies that the above system has a unique solution.

## 4 Modifications for non-critical bundles over $\mathbb{CP}^n$ .

So far, our discussion has been focusing on critical bundles. For non-critical bundles  $V$  over  $\mathbb{CP}^n$ ,  $\text{rank}(U_d)$  and  $\dim \overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d)$  are different for some  $d$ , by definition. If one takes  $\{Q_d\}_d$  associated to the top Chern class of  $V$ , then one will simply get 0 for  $K_d$ . Let  $r$  be the rank of  $V$ . Then, to obtain more interesting invariants, one may consider taking  $\{Q_d\}_d$  to be the Euler data associated to the Chern polynomial

$$c(x) = x^r + c_1(V)x^{r-1} + \cdots + c_{\text{top}}(V)$$

of  $V$ . In this case, some details in Sec. 2 and Sec. 3 will have to be modified accordingly; however, the conceptual flow remains the same.

The various items that are involved in the actual computation of  $Q_d$  and  $K_d$  and their modifications following [L-L-Y] are listed below:

- **degree bound of  $Q_d$ :** Since the top Chern class gives the highest degree terms, both the total degree bound and the  $\alpha$ -degree bound remain valid.
- **the special values  $i_{p_{i,0}}^*(Q_d)\big|_{\alpha=\frac{\lambda_i-\lambda_j}{d}}$ :** These special values are now given by

$$i_{p_{i,0}}^*(Q_d)\big|_{\alpha=\frac{\lambda_i-\lambda_j}{d}} = \prod_a \prod_{m=0}^{l_a d} \left(x + l_a \lambda_i - m \frac{\lambda_i - \lambda_j}{d}\right) \prod_b \prod_{m=1}^{k_b d-1} \left(x - k_b \lambda_i + m \frac{\lambda_i - \lambda_j}{d}\right).$$

This leads to a corresponding change in the computation of  $Q_1$  and right-hand-side of the third linear systems in Sec. 3. The first, second, and the fourth linear systems remains valid.

- **from  $Q_d$  to  $K_d$ :** Let  $s = \text{rank}(U_d) - \dim \overline{\mathcal{M}}(\mathbb{CP}^n, d)$ . Theorem 7.2 in [L-L-Y, II] gives

$$\frac{1}{s!} \left. \frac{d^s}{dx^s} \right|_{x=0} \int_{\mathbb{CP}^n} e^{-Ht/\alpha} \frac{\lim_{\lambda \rightarrow 0} I_d^*(Q_d)}{\prod_{m=1}^d (H - m\alpha)^{n+1}} = \frac{1}{\alpha^3 x^s} (2 - dt) K_d.$$

Note that, for the validity of this formula alone, it is not required that  $s_d$  be independent of  $d$  [Li]. Thus, once  $Q_d$  is obtained by solving the system of linear equation,  $K_d$  follows.

Based on Sec. 3 and the discussion here, a code `EulerData_MP.m` is written. The detail is in Sec. 6.

## 5 Examples.

Using the Maple code "EulerData\_MP.m" in Sec. 6 and taking  $\Omega$  to be the Chern polynomial, we compute the first few  $K_d$  for some non-critical bundles, as listed in TABLE 5-1 (cf. Cases 6, 14-18 in SEC. 3 of the code).

bundle	$d = 1$	$d = 2$	$d = 3$
$T_*\mathbb{CP}^2$	$10x^2$	—	—
$O_{\mathbb{CP}^4}(6)$	$50400x$	$(752729895/4)x^2$	$(433244745198080/243)x^3$
$O_{\mathbb{CP}^4}(7)$	$451570x^2$	$(403985396325/32)x^4$	$(15755269694706695755/17496)x^6$
$O_{\mathbb{CP}^4}(8)$	$2773820x^3$	$(3178734062035/8)x^6$	$(46028387589557254161275/314928)x^9$
$O_{\mathbb{CP}^4}(9)$	$13198850x^4$	$(243281907041715/32)x^8$	$(197802281929974511821535/17496)x^{12}$
$O_{\mathbb{CP}^4}(10)$	$52040450x^5$	$(25908993204089625/256)x^{10}$	$(71418501571607082433686025/139968)x^{15}$

TABLE 5-1.  $K_d$  for some non-critical bundles.

Note that, in TABLE 5-1, if one formally converts  $K_d$  to  $n_d$  using the usual multiple cover formula  $K_d = \sum_{k|d} n_d \frac{1}{k} \frac{1}{k^3}$ , then the  $n_d$  thus obtained will no longer be integers for  $d \geq 2$ . Compared with the other twelve examples computed/tested using the code, this indicates that for those non-critical bundles over  $\mathbb{CP}^n$ , whose  $\text{rank}(U_d) - \dim(\overline{\mathcal{M}}_{0,0}(\mathbb{CP}^n, d))$  depends on  $d$  in a non-trivial way, the above multiple cover formula will have to be modified. Exactly how is an issue for further investigation.

## 6 A package in Maple V for the computation of $Q_d$ and $K_d$ .

### General remarks.

A few remarks are given below concerning the code, its current scope, and its usage.

- Two bad things first. The first one is that the code, as currently written, is limited only to the case that  $\Omega$  is an Euler class or a Chern polynomial. For other multiplicative characteristic classes, one will not be able to use the code to make sensible computation without entering the core part, SEC. 1.3 (Q1K1) and SEC. 2 (Euler-Data), to do some modifications. This shortage will hopefully be gradually removed along with the development of the theory, using the relation of a given multiplicative class with Chern roots. The second one is that the actual computation of Euler data and  $K_d$  could be a very demanding task both for Maple V and the machine (cf. some words in SEC. 3 of the code about Test 6:  $T_*\mathbb{CP}^2$  even for  $d = 2$ ). When it exceeds the capacity the Maple V, one will likely get an error message. Experience tells us that occasionally these messages may be misleading.
- Now let us mention something more positive. As indicated in SEC. 3 of the code, the code has been tested correct for all known cases within the capacity of Maple V and the machine used. It is also made user-friendly: *to run for a case of study, one only has to follow the examples in SEC. 3 and modify the various arguments/parameters to be fed into the function ‘EulerData’, as instructed there.* By no means does one need to do anything else; *nor* do we assume any knowledge of Maple at all.
- The specialization used, the time consumed, and the RAM memory used for the examples tested, particularly those that take long hours, are recorded in SEC. 3 of the code for reference. Only the last run of each case is recorded in these notes.

### Instruction of running the code under Window 98.

For non-maple-user, let us give here some instruction of running the code under Window 98: (assuming there is already Maple V in the folder “Programm Files”)

- Read first the instructions both at the beginning of the code and at the start of SEC. 3 of the code .
- Save the code (say by the file name ‘EulerData\_MP.m’) in the subfolder ‘Bin.wnt’ in the folder ‘Maple V Release 5’ that is automatically set up in the folder ‘Programm Files’ when installing Maple V. Double click the maple icon to open a Maple V worksheet. Inside the worksheet, type in the following command line after the prompt

```
restart; read "EulerData_MP.m";
```

then hit Enter on the keyboard. The output will be displayed directly on the worksheet.

For all other operating systems, please consult the system manager.

## The code 'EulerData\_MP.m'.

```

===== ( Beginning of the code. )=====

#----- ( PLEASE READ ME BEFORE USE )-----
# NOTE 1. Though the code has been thoroughly tested, nevertheless if you find any bug that has escaped
# our scrutiny, please send an e-mail to 'chienliu@math.harvard.edu' for remedy of this.
#
# NOTE 2. CAUTION TO USER:
# The machine used for the tests is
#
# Dell PC with Pentium II 400 MH, Cache 512 KB, RAM 384 MB, Hard Drive 7 GB.
#
# Make sure the code is run in a machine at least of the same level as above to avoid possible crashes.
#
# NOTE 3. Anything free comes with no guarantee. Please take your own caution/judgement or consult experts
# BEFORE running the code. The author of the code shall not be responsible for any possible damage
# and/or loss caused by the code. The author does not recommend the use of the code for applications
# in which errors or omissions could threaten life, injury or significant loss.
=====

#-----
# Language used: Maple V, created by Waterloo Maple Inc. at Waterloo, Ontario, Canada.
#-----

#-----
# SEC. 0 Introduction and outline.
#-----

#-----
# EulerData_MP.m -- a package in Maple V.
# Given an Euler class or the Chern polynomial of a concavex bundle  $V$  over  $CP^n$ , the
# package computes the Euler data  $Q=\{Q_d\}$  and the intersection numbers  $K_d$  for  $V$ 
#-----
# Date of completion: November 21, 1999.
# Test: All the subroutines involved are tested correct separately. See the individual remarks for
# details. Some records of the run are recorded in SEC. 3 for a reference of future improvements.
# Date of last revision: January 7, 2000.
#-----
# List of functions/subroutines:
# Main routine (Sec. 2): EulerData.
# Subroutines (Sec. 1): LSolveSub (Sec. 1.1.1), LSolveSub_A (Sec. 1.1.2 ),
# PolyCongruent (Sec.1.2), Q1K1 (Sec.1.3).
#-----
# SEC.3 contains INSTRUCTIONS of using the code and various cases of computation and serves as
# a self-explanatory TUTORIAL for users.
# NOTE. For the computataion of  $K_d$  alone, one may add a specialization BEFORE employing the function
# 'EulerData'. Different cases or different degrees for the same case may require different
# specializations. To be compatible with the code, the specialization must be of the form in the
# variables 'lambda' and 'u'. The specialization is given in the form:
#
# lambda :i -> some_made_up_function(i, u)
#
# such that lambda(i)=0 when u=0. (See the examples in SEC. 3.)
#-----

#-----
# SEC. 1 Subroutines.
#-----

```



```

#-----
# SEC. 1.1.1 Definition of the function 'LSolveSub'.
#-----
# 'LSolveSub(equation_set, variable_list, check)'
# solves a system 'equation_set' of linear equations in variables in 'variable_list' by
# eliminations and back substitutions.
#
# NOTE: If 'check'=0 (resp. 1), then the consistency of the system will not (resp. will) be checked.
#-----
# Date of completion: November 19, 1999.
# Test: Tested correct for six sample examples.
# Date of last revision: November 23, 1999.
#-----

LSolveSub := proc( equation_set, variable_list, check )
local eqn_list_diag, eqn_list_tri, eqn_set, i, j, list_part,
      m, modification, s, sol, t, test, v, vl:

eqn_set := equation_set:
eqn_set := map( el->lhs(el)-rhs(el)=0, eqn_set ):
m := nops(eqn_set):
v := variable_list:
vl := nops(v):
eqn_list_tri := []:

# print("Number of equations = ", m):
# print("Number of variables = ", vl):

# print("LSolveSub: 1000000"):

for i from 1 to vl do
#   print("i = ", i):

  for j from 1 to m do
#     print("j = ", j):
t := j:
test := coeff( lhs(eqn_set[j]), v[i] ):
#   print("v[i] = ", v[i]): #####
#   print("LSolveSub: 1220000"): #####
if test<>0 then
#     print("test = ", test): #####
#     print("eqn_set[j] = ", eqn_set[j]): #####
#     print("v[i] = ", v[i]):
s := solve( { eqn_set[j] }, { v[i] } ):
#     print("s = ", s):
sol := s[1]:
fi:
#   print("LSolveSub: 1240000"): #####
if test<>0 then break fi:
od:

# print("LSolveSub: 1400000"):

eqn_list_tri := [ op(eqn_list_tri), sol ]:
eqn_set := eqn_set minus { eqn_set[t] }:

if nops(eqn_set)=0 then break fi:

eqn_set := map( el -> subs( sol, el ), eqn_set ):
eqn_set := map( el -> expand( lhs(el) )=0, eqn_set ):
eqn_set := eqn_set minus {0=0}:

```

```

    if eqn_set <> {} then
      eqn_set := map( el -> collect( lhs(el), v )=0, eqn_set )
    fi:

    m := nops( eqn_set ):
  od:

#-----
# In terms of matrices, so far an upper triangular matrix is formed.
# The next step is the back substitutions to solve v.
#-----

# print("LSolveSub: 6000000"):

m := nops( eqn_list_tri ):
eqn_list_diag := eqn_list_tri:

unassign( 'eqn_list_tri' ):

for i from 2 to m do
  list_part := [ op(-(i-1)..-1, eqn_list_diag)]:
  modification := rhs( op( -i, eqn_list_diag ) ):
  modification := subs( list_part, modification ):
  modification := normal( modification ):
  modification := simplify( modification ):
  eqn_list_diag := subsop( [-i,2]=modification, eqn_list_diag ):
od:

#-----
# consistency check.
#-----

# print("LSolveSub: 9000000");

if check=1 then
  if eqn_set<>{} then
    eqn_set = map( el -> subs( eqn_list_diag, el), eqn_set ):
    eqn_set = { map( el -> lhs(el)-rhs(el), eqn_set ) }:
    if eqn_set <> {0} then
      print("inconsistent")
    else print("consistent")
    fi:
  else print("consistent")
fi:

unassign( 'eqn_list_tri', 'eqn_set', 'i', 'j', 'list_part',
          'm', 'modification', 's', 'sol', 't', 'test', 'v', 'vl'):
RETURN(eqn_list_diag):
end:

#-----
# SEC. 1.1.2 Definition of the function 'LSolveSub_A'.
#-----
# 'LSolveSub_A(equation_set, variable_list, check)'
# solves a system 'equation_set' of linear equations in variables in 'variable_list'
# by eliminations and back substitutions.
#
# NOTE: If 'check'=0 (resp. 1), then the consistency of the system will not (resp. will) be checked.
#-----

```

```

# Date of completion: November 19, 1999.
# Test: Tested correct for six sample examples.
# Date of last revision: January 6, 1999.
# Modification from 'LSolveSub':
# Replacing the built-in function 'solve' by direct algebra to increase the capacity of the code.
#-----

LSolveSub_A := proc( equation_set, variable_list, check )
local eqn_list_diag, eqn_list_tri, eqn_set, i, j, list_part,
    m, modification, s, s_1, sol, t, test, v, vl:

eqn_set := equation_set:
eqn_set := map( el->lhs(el)-rhs(el)=0, eqn_set ):
m := nops(eqn_set):
v := variable_list:
vl := nops(v):
eqn_list_tri := []:

# print( "Number of equations = ", m ):
# print( "Number of variables = ", vl ):

# print( "LSolveSub_A: 1000000" ):

for i from 1 to vl do
#   print( "i = ", i ):

    for j from 1 to m do
#       print( "j = ", j ): #####
#       ( "LSolveSub_A: 1220000" ): #####
        t := j:
        test := coeff( lhs(eqn_set[j]), v[i] ):
#       print( "v[i] = ", v[i] ):
        if test<>0 then
#           print( "test = ", test ): #####
#-----
# Remark: Replacement of the following line in 'LSolveSub'.
#-----
#       s := solve( { eqn_set[j] }, { v[i] } ):
#-----
#           print( "v[i] = ", v[i] ):
#           s_1 := -(lhs(eqn_set[j]) -test*v[i])/test:
#           s_1 := normal(s_1):
#           s := {v[i]=s_1}:
#           print( "s = ", s ):
#-----
# End of replacement.
#-----
            sol := s[1]:
            fi:
            if test<>0 then break fi:
        od:

#       print( "LSolveSub_A: 1400000" ):

        eqn_list_tri := [ op(eqn_list_tri), sol ]:
        eqn_set := eqn_set minus { eqn_set[t] }:

        if nops(eqn_set)=0 then break fi:

    eqn_set := map( el -> subs( sol, el ), eqn_set ):
    eqn_set := map( el -> expand( lhs(el) )=0, eqn_set ):
    eqn_set := eqn_set minus {0=0}:

```

```

    if eqn_set <> {} then
      eqn_set := map( el -> collect( lhs(el), v )=0, eqn_set )
    fi:

    m := nops( eqn_set ):
  od:

#-----
# In terms of matrices, so far an upper triangular matrix is formed.
# The next step is the back substitutions to solve v.
#-----

# print( "LSolveSub_A: 6000000" ):

m := nops( eqn_list_tri ):
eqn_list_diag := eqn_list_tri:

for i from 2 to m do
  list_part := [ op(-(i-1)..-1, eqn_list_diag)]:
  modification := rhs( op( -i, eqn_list_diag ) ):
  modification := subs( list_part, modification ):
  modification := normal( modification ):
  modification := simplify( modification ):
  eqn_list_diag := subsop( [-i,2]=modification, eqn_list_diag ):
od:

#-----
# consistency check.
#-----

# print("LSolveSub_A: 9000000"):

if check=1 then
  if eqn_set<>{} then
    eqn_set = map( el -> subs( eqn_list_diag, el), eqn_set ):
    eqn_set = { map( el -> lhs(el)-rhs(el), eqn_set ) }:
    if eqn_set <> {} then
      print("inconsistent")
    else print("consistent")
    fi:
  else print("consistent")
  fi:
fi:

unassign( 'eqn_list_tri', 'eqn_set', 'i', 'j', 'list_part',
          'm', 'modification', 's', 's_1', 'sol', 't', 'test', 'v', 'vl'):

RETURN(eqn_list_diag):
end:

#-----
# SEC. 1.2 Definition of the function 'PolyCongruent'.
#-----
# 'PolyCongruent(polynomial, replacement, variable)'
# gives the polynomial of minimal degree in 'variable' that is congruent to 'polynomial'
# modulo the relation 'replacement'.
#
# Note: The argument 'replacement' must be of the form:
#
# variable^n = polynomial in 'variable' of degree <= n

```

```

#
#-----
# Date of completion: November 20, 1999.
# Test: Tested correct for sample examples.
# Date of last revision: November 20, 1999.
#-----

PolyCongruent := proc( polynomial, replacement, variable )
local d0, d1, dnow, i, relation, relation_set, poly, v:

poly := polynomial:
relation := replacement:
v := variable:
d0 := degree( lhs(relation), v ):
d1 := degree( poly, v ):

relation_set :=
{ seq( v^(i+d0)=collect( v^i*rhs(relation), v ), i=0..d1-d0 ) }:

dnow := d1:
for i from 1 to d1-d0+1 do
if dnow < d0 then break fi:
poly := subs( relation_set, poly ):
poly := expand( poly ):
poly := simplify( poly ):
poly := collect( poly, v ):
dnow := degree( poly, v ):
od:

unassign( 'd0', 'd1', 'dnow', 'i', 'relation', 'relation_set', 'v' ):
RETURN(poly):
end:

#-----
# SEC. 1.3 Definition of the function 'Q1K1'.
#-----
# 'Q1K1(n, splitting_type, s_diff_0, opt_1)'
# gives the first term in the  $\Omega$ -Euler data and the first Kontsevich number for a concavex
# bundle  $EV$  over  $\mathbb{C}P^n$ , where  $\Omega$  is the Chern polynomial.
#
# Note.1: The argument 'splitting_type' must be of the form "[_, _, ...],[_, _, ...]", where the
# first [_, _, ...] comes from the convex summand and the second [_, _, ...] comes from the
# concave summand of the concavex bundle  $EV$ .
#
# Note.2: The argument 'opt_1' takes only values 0 and 1.
# If 'opt_1' = 0, then only  $K_1$  will be computed.
# If 'opt_1' = 1, then exact  $Q_1$  will also be computed.
#-----
# Date of completion: November 19, 1999.
# Test: The value  $K_1$  are correct for nine known cases: Cases 1 - 5, 7-10 in SEC. 3.
# Date of last revision: November 19, 1999.
#-----

Q1K1 := proc(n, splitting_type, s_diff_0, opt_1)
local i, inst1, k, k1, k1a, l1, l2, poly, poly1,
qipoly0, qipoly01, qipoly02, qipoly10, qipoly11, qipoly110, qipoly111,
qisummand1, qisummand2, q1value, q1value1, q1value2,
q1, q10, q10a, q11, q11a, s1, s2, v, y:

l1 := splitting_type[1]:
l2 := splitting_type[2]:

```

```

s1 := nops(l1):
s2 := nops(l2):

print("l1 = ", l1):
print("l2 = ", l2):
print("s1 = ", s1):
print("s2 = ", s2):

print("Q1K1, 1000000"):

q1value1 := proc(v, y)
  local a, m:
  if s1<>0 then
    mul( mul(x+l1[a]*v-m*y, m=0..l1[a]), a=1..s1 )
  else 1
  fi:
end:

q1value2 := proc(v, y)
  local b, m:
  if s2<>0 then
    mul( mul(x+l2[b]*v+m*y, m=1..-l2[b]-1), b=1..s2 )
  else 1
  fi:
end:

q1value := (v, y) -> q1value1(v, y)*q1value2(v, y):

q1summand1 := proc(i, j, alpha)
  local factor1, factor2, k:
  factor1 := 1:
  for k from 0 to n do
    if k<>i and k<>j then
      factor1 := factor1*( alpha-lambda(i)+lambda(k) )
    else factor1 := factor1
    fi:
  od:
  factor2 := 1:
  for k from 0 to n do
    if k<>i and k<>j then
      factor2 := factor2*( lambda(k)-lambda(j) )
    else 1
    fi:
  od:
  q1value( lambda(i), lambda(i)-lambda(j) )*factor1/factor2:
end:

q1summand2 := (i, j, alpha) ->
  normal( q1summand1(i, j, alpha) ):

#-----
# THEORY: 'q1poly01(i, alpha)' =  $i^{\ast}_{p_{i,0}}(Q_1)$ 
#-----

q1poly01 := proc(i, alpha)
  local j, t:
  t := 0:
  for j from 0 to n do
    if j<>i then
      t := t + q1summand2(i, j, alpha)
    fi:
  od:
end:

```

```

        else t := t
      fi:
    od:
end:

qipoly02 := (i, alpha) -> normal( qipoly01(i, alpha) ):
qipoly0 := (i, alpha) -> simplify( qipoly02(i, alpha) ):

qipoly10 := (i, alpha) -> normal( qipoly0(i, alpha) ):
qipoly110 := (i, alpha) -> simplify( qipoly10(i, alpha) ):
qipoly11 := (i, alpha) -> qipoly10(i, -alpha):
qipoly111 := (i, alpha) -> simplify( qipoly11(i, alpha) ):

#-----
# definition of q10
#-----

q10a := proc(n)
  local i, q10_factor2, q10_factor3, total:

  q10_factor2 := proc(i, n)
    local j, k, t:
    t := 1:
    for j from 0 to n do
      for k from 0 to 1 do
        if j<>i or k<>0 then
          t := t*(kappa-lambda(j)-k*alpha)
        else t:= t
        fi:
      od:
    od:
  end:

  q10_factor3 := proc(i, n)
    local j, k, t:
    t := 1:
    for j from 0 to n do
      for k from 0 to 1 do
        if j<>i or k<>0 then
          t := t*( lambda(i)-lambda(j)-k*alpha )
        else t := t
        fi:
      od:
    od:
  end:

  total := add( qipoly110(i,alpha)*q10_factor2(i,n)/q10_factor3(i,n),
    i=0..n ):
end:

q10 := normal( q10a(n) ):
q10 := simplify( q10 ):

#-----
# defintion of q11
#-----
print("Q1K1, 4000000"):

q11a := proc(n)
  local i, q11_factor2, q11_factor3, total:
  q11_factor2 := proc(i, n)
    local j, k, t:

```

```

t := 1:
for j from 0 to n do
  for k from 0 to 1 do
    if j<>i or k<>1 then
      t := t*( kappa-lambda(j)-k*alpha )
    else t := t
  fi:
od:
od:
end:

q11_factor3 := proc(i, n)
  local j, k, t:
  t := 1:
  for j from 0 to n do
    for k from 0 to 1 do
      if j<>i or k<>1 then
        t := t*( lambda(i)-lambda(j)+(1-k)*alpha )
      else t := t
    fi:
  od:
od:
end:

total := add( q1poly111(i,alpha)*q11_factor2(i, n)/q11_factor3(i,n),
              i=0..n ):
end:

q11 := normal( q11a(n) ):
q11 := simplify( q11 ):

print("Q1K1, "):

#-----
# The first term of Euler data : Q1.
#-----

q1 := q10+q11:
q1 := normal(q1):
q1 := simplify(q1):

print("Q1K1, 8000000"):

#-----
# Computation of K1 from Q1, using the formula in Lian, Liu, and Yau's "Mirror Principle I, II"
# and [Cox-Katz] "Mirror symmetry and algebraic geometry".
#-----

poly := q1:
if opt_1=0 then
  poly := subs( u=0, q1 )
else
  for i from 0 to n do
    poly := subs( lambda(i)=0, poly )
  od:
fi:

poly := subs( kappa=h, poly ):

#-----
# Extra manipulation for noncritical bundles with Omega=the Chern polynomial.
#-----

```



```

if s_diff_0(1) <> 0 then
  poly := diff( poly, x$s_diff_0(1) ):  #( Cf. Mirror Principle II, p. 36. )
  poly := subs( x=0, poly )
fi:

#-----

poly1 := add( (-h*t/alpha)^k/(k!), k=0..n )*(-1)^(n+1)/(alpha^(n+1))*
  (add( (h/alpha)^k, k=0..n))^(n+1)*poly:
poly1 := simplify(poly1):

k1 := alpha^3/(2-t)*coeff(poly1,h^n):
k1 := normal(k1):
inst1 := k1:

#-----
# Extra manipulation with Omega=the Chern polynomial
#-----

if s_diff_0(1) <> 0 then
  inst1 := k1/(s_diff_0(1)!):
  k1 := x^(s_diff_0(1))*k1/(s_diff_0(1)!)
fi:

print("Routine: Q1K1"):
print("k1 = ", k1):
print("inst1 = ", inst1):

#-----
# k1a := alpha^2*coeff(h*poly1, h^n):
# k1a := normal(k1a):
#-----

unassign('i', 'k', 'l1', 'l2', 'poly', 'poly1', 'q1poly0', 'q1poly01',
  'q1poly02', 'q1poly10', 'q1poly11', 'q1poly110', 'q1poly111',
  'q1summand1', 'q1summand2', 'q1value', 'q1value1', 'q1value2',
  'q10', 'q10a', 'q11', 'q11a', 's1', 's2', 'v', 'y' ):

RETURN( [q1, k1, inst1] ):
# RETURN([q1, k1, k1a]):
end:

#-----
# SEC. 2 The MAIN ROUTINE.
#-----

#-----
# Definition of the function 'EulerData'.
#-----
# 'EulerData(n, splitting_type, Omega_0, s_diff_0, dmax, opt_1, opt_2, check)'
# computes $Q_1, \dots, Q_d$ with $Q_0=\Omega_0$ and $K_1, \dots, K_d$ for a convex bundle $V$ over $\mathbb{CP}^n$
# of splitting type 'splitting_type'.
#
# Note.1: The argument 'splitting_type' must be in the form of lists "$[[_1, _2, \dots], [_3, _4, \dots]]$", where
# the first $[_1, _2, \dots]$ comes from the convex summand and the second $[_3, _4, \dots]$ comes from the
# concave summand of the concavex bundle $V$.
#
# Note.2: The argument 's_diff_0' is the difference: $\text{rank}(U_d) - \dim(\overline{\text{cal M}}_{0,0}(\mathbb{CP}^n, d))$;
# (cf. [L-L-Y1, 2] for notations).
#

```

```

# Note.3: The argument 'opt_1' takes only values 0 and 1.
#         If 'opt_1' = 0, then only $K_d$, $d=1, ..., dmax$, will be computed.
#         If 'opt_1' = 1, then exact $Q_d$ up to $Q_{dmax}$ will also be computed.
#
# Note.4: The argument 'opt_2' takes only values 0 and 1.
#         If 'opt_2' = 0, then 'LSolveSub' is employed;
#         if 'opt_2' = 1, then 'LSolveSub_A' is employed.
#
# Note.5: The argument 'check' takes only values 0 and 1.
#         If 'check' = 0 (resp. 1), then consistency check of the linear system will not (resp. will)
#         be performed.
#-----
# Date of completion: November 21, 1999.
# Test: See SEC. for details.
# Date of last revision: December 8, 1999.
#-----

EulerData := proc(n, splitting_type, Omega_0, s_diff_0, dmax, opt_1, check)
local d, eqn_set, inhomogen11, inhomogen12, inhomogen13, inhomogen1,
      inhomogen3, inhomogen31, inhomogen32, instanton_list,
      k, k1, k1a, ka, kd, kda, kdn, l1, l2, n1,
      poly, poly1, poly_set, poly_set1, poly_set11, poly_set21,
      poly_set2, poly_set3, poly_set4, q, q0, q1, q1_difference,
      qd, qd11, qd111, qd112,
      qd113, qd114, qd12, qd13, qd21, qd22, qd23, qd31, qd311, qd312, qd313,
      qd32, qdr, qdr1, qd_value, qd_value1, qd_value2, qk1, qk1_linear,
      rel, s1, s2, s_diff, substitution, variable_list:

qk1 := Q1K1(n, splitting_type, s_diff_0, opt_1):

print("d=1"):
print("k1 = ", qk1[2]):
# print("k1 =", qk1[2], qk1[3]):

print("x = ", x):
print("Routine: EulerData; Omega = ", Omega):

if dmax=1 then RETURN(qk1) fi:

#-----
# Computations of $Q_d$ and $K_d$ for $d \ge 2$.
#-----

l1 := splitting_type[1]:
l2 := splitting_type[2]:
s1 := nops(l1):
s2 := nops(l2):

q0_inverse := 1/Omega_0:
print("Routine: EulerData; q0_inverse = ", q0_inverse):

q1 := qk1[1]:
q := [ q1 ]:
k1 := qk1[2]:
k := [ k1 ]:
# k1a := qk1[3]:
# ka := [ k1a ]:
instanton_list := [ qk1[3] ]:

print("0000000000000000"):
print("k = ", k):
# print("ka = ", ka):

```

```

print("instanton_list = ", instanton_list):

for d from 2 to dmax do
  print("d = ", d):
  n1 := (n+1)*d+n-3:
  qd := add( add( w(mu,nu)*alpha^mu*kappa^nu, nu=0..n1-mu ), mu=0..n1 ):
  variable_list := [ seq( seq( w(mu, nu), nu=0.. n1-mu ), mu=0..n1 ) ]:

  #-----
  # Linear system from the 1st set of equations.
  #-----
  print("MAIN: 1000000; I am now working out the first set of equations."):

  qd11 := (i, r) -> subs( kappa=lambda(i)+r*alpha, qd ):
  qd112 := (i, r) -> expand( qd11(i, r) ):
  qd113 := (i, r) -> simplify( qd112(i, r) ):
  qd114 := (i, r) -> collect( qd113(i,r), alpha ):

  inhomogen11 := (i, r) ->
    subs( { kappa=lambda(i), alpha=-alpha}, q[r] ) *
    subs( kappa=lambda(i), q[d-r] ) * subs( h=lambda(i), q0_inverse ):
  inhomogen12 := (i, r) -> normal( inhomogen11(i, r) ):
  inhomogen13 := (i, r) -> simplify( inhomogen12(i, r) ):
  inhomogen1 := (i, r) -> collect( inhomogen13(i, r), alpha ):

  qd11 := (i, r) -> qd114(i, r)-inhomogen1(i, r):
  qd12 := (i, r) -> simplify( qd11(i, r) ):
  qd13 := (i, r) -> collect( qd12(i, r), alpha ):
  poly_set11 := (i, r) -> coeffs( qd13(i, r), alpha ) :
  poly_set1 := { seq( seq( poly_set11(i, r), r=1..d-1 ), i=0..n ) }:
  poly_set1 := map( el -> collect( el, variable_list ), poly_set1 ):

  #-----
  # Linear system from the 2nd set of equations.
  #-----
  print("MAIN: 2000000; I am now working the second set of equations."):

  qd21 := i -> subs( kappa=lambda(i)+d*alpha, qd ) -
    subs( { kappa=lambda(i), alpha=-alpha }, qd ):
  qd22 := i -> simplify( qd21(i) ):
  qd23 := i -> collect( qd22(i), alpha ):
  poly_set21 := i -> coeffs( qd23(i), alpha ):
  poly_set2 := { seq( poly_set21(i), i=0..n ) }:
  poly_set2 := map( el -> collect( el, variable_list ), poly_set2 ):

  #-----
  # Linear system from the 3rd set of equations:
  # the special values $Q_d(\lambda_i, (\lambda_i-\lambda_j)/d )$
  #-----
  print("MAIN: 3000000; I am now working out the third set of equations."):

  qd_value1 := proc(v, y)
  local a, m, t:
  if s1<>0 then
    t := mul( mul( x+l1[a]*v-m*y, m=0..l1[a]*d ), a=1..s1 )
    else t := 1
  fi:
  RETURN(t):
end:

qd_value2 := proc(v, y)
local b, m, t:

```

```

if s2<>0 then
  t := mul( mul( x+12[b]*v+m*y, m=1..-12[b]*d-1 ), b=1..s2 )
  else t := 1
fi:
RETURN(t):
end:

qd_value := (v, y) -> qd_value1(v, y)*qd_value2(v, y):
inhomogen31 := (i, j) -> qd_value( lambda(i), (lambda(i)-lambda(j))/d ):
inhomogen32 := (i, j) -> normal( inhomogen31(i,j) ):
inhomogen3 := (i, j) -> simplify( inhomogen32(i, j) ):
qd311 := (i, j) ->
  subs( { kappa=lambda(i), alpha=( lambda(i)-lambda(j) )/d }, qd):
qd312 := (i, j) -> expand( qd311(i, j) ):
qd313 := (i, j) -> simplify( qd312(i, j) ):
qd31 := (i, j) -> qd313(i, j)-inhomogen3(i, j):
qd32 := (i, j) -> simplify( qd31(i, j) ):
poly_set3 := { seq( seq( qd32(i, j), j=i+1..n), i=0..n ) }:
poly_set3 := map( el-> collect( el, variable_list ), poly_set3 ):

#-----
# Linear equations from the 4th set of equations:
#  $\alpha$ -degree bound:  $\deg_{\alpha} I^{\ast}_d(Q_d) \leq (n+1)d-2$ .
#-----
print("MAIN: 4000000; I am now working out the fourth set of equations."):

rel := mul( kappa-lambda(i), i=0..n ):
rel := expand( rel )-kappa^(n+1):
rel := collect( rel, kappa ):
rel := kappa^(n+1)=rel:

qdr := PolyCongruent( qd, rel, kappa ):
qdr := collect( qdr, alpha ):
poly_set4 := { seq( coeff( qdr, alpha^i ), i=(n+1)*d-1..n1 ) }:
poly_set4 := map( el -> collect( el, kappa ), poly_set4 ):
poly_set4 := map( el -> coeffs( el, kappa ), poly_set4 ):

#-----
# Combination of the four linear systems and solve the system.
#-----
print("MAIN: 6000000; I am now combining the four systems of equations."):

poly_set := 'union'( poly_set1, poly_set2, poly_set3, poly_set4);
eqn_set := map( el -> el=0, poly_set );

unassign('poly_set', 'poly_set1', 'poly_set2', 'poly_set3', 'poly_set4'):

##-----
## Added for testing the consistency check routine.
##
## print( "op(-1, eqn_set) = ", op(-1, eqn_set) ):
### eqn_set := { op( eqn_set ), lhs( op(-1, eqn_set) )=100 }: # or
### eqn_list := { op( eqn_set ), 2*lhs( op(-1, eqn_set) )=0 }:
##
## End of testing lines
##-----

if opt_2 = 0 then
  print("I am now solving the system of linear equations using 'LSolveSub'."):
  substitution := LSolveSub( eqn_set, variable_list, check )
else
  print("I am now solving the system of linear equation using 'LSolveSub_A'."):

```

```

        substitution := LSolveSub_A( eqn_set, variable_list, check )
fi:

print("substitution = ", substitution):
qd := subs( substitution, qd );
qd := normal(qd);
qd := simplify(qd);
q := [ op(q), qd ];
print("The system of linear equations is now solved."):
print("q = ", q):

#-----
# The nonequivariant limit of  $I_d^{\ast}(Q_d)$ .
#-----
print("I am now turning to compute  $Q_d$ ,  $K_d$ , and  $n_d$ ."):

qdr1 := PolyCongruent( qd, rel, kappa ):

if opt_1=0 then poly := subs( u=0, qdr1 )
else poly := qdr1:
    for i from 0 to n do
        poly := subs( lambda(i)=0, poly )
    od:
fi:
poly := subs( kappa=h, poly ):

#-----
# Extra manipulation for noncritical bundles with  $\Omega$ =the Chern polynomial.
# (Cf. Mirror principle, I: p. 37.)
#-----

if s_diff_0(d) <> 0 then
    poly := diff( poly, x$s_diff_0(d) ):
    poly := subs( x=0, poly )
fi:

#-----
# Computation of  $K_d$  via  $Q_d$ , using the formula in
# [Lian-Liu-Yau] "Mirror principle I", "Mirror principle II".
#-----

poly1 := add( (-h*t/alpha)^r/(r!), r=0..n ) *
    mul( ( add( (h/(m*alpha))^r, r=0..n ) )^(n+1), m=1..d ) *
    poly/( (d!)^(n+1)*(-alpha)^(d*(n+1)) ):
poly1 := normal(poly1):
poly1 := simplify(poly1):
poly1 := collect(poly1, h):
kd := alpha^3/(2-d*t)*coeff(poly1, h^n):
kd := normal(kd):
kdn := kd:

#-----
# Extra manipulation for noncritical bundles with  $\Omega$ =the Chern polynomial.
#-----

if s_diff_0(d) <> 0 then
    kdn := kd/(s_diff_0(d)!):
    kd := x^(s_diff_0(d))*kd/(s_diff_0(d)!):
    kd := normal(kd)
fi:

print("kdn = ", kdn):

```

```

print("kd = ", kd):

#-----

k := [ op(k), kd ]:
print("k = ", k):

#-----
# Computation of $K_d$ via $Q_d$, using a formula in
# Cox and Katz: Mirror symmetry and algebraic geometry
#-----
#
# kda := alpha^2*coeff( h*poly1, h^n )/d:
# kda := normal(kda):
# ka := [ op(ka), kda ]:
# print("ka = ", ka):

#-----
# Computation of instanton numbers from intersection numbers.
#-----

InstantonN := proc( inst_list, k_d )
local i, l, nd, new_inst_list, s:

l := nops( inst_list ):
nd := k_d:

for i from 2 to l+1 do
  if irem(l+1, i)=0 then
    s := iquo(l+1, i):
    nd := nd - inst_list[s]/i^3:
  fi:
od:

new_inst_list := [ op(inst_list), nd ]:

unassign( 'i', 'l', 'nd', 's' ):
RETURN(new_inst_list):
end:

instanton_list := InstantonN( instanton_list, kdn):
print("instanton_list = ", instanton_list):

od:

unassign( 'd', 'eqn_set', 'inhomogen11', 'inhomogen12', 'inhomogen13',
'inhomogen1', 'inhomogen3', 'inhomogen31', 'inhomogen32',
'k1', 'k1a', 'kd', 'kda', 'kdn', 'l1', 'l2', 'n1', 'poly', 'poly1',
'poly_set', 'poly_set1', 'poly_set11', 'poly_set21', 'poly_set2',
'poly_set3', 'poly_set4', 'q0_inverse', 'q1', 'q1_difference',
'qd', 'qd11', 'qd111', 'qd112', 'qd113', 'qd114', 'qd12', 'qd13',
'qd21', 'qd22', 'qd23', 'qd31', 'qd311', 'qd312', 'qd313', 'qd32',
'qdr', 'qdr1', 'qd_value', 'qd_value1', 'qd_value2', 'qk1',
'qk1_linear', 'rel', 's1', 's2', 's_diff', 'screen', 'substitution',
'variable_list' ):

RETURN([q, k, instanton_list]):
# RETURN([q, k, ka, instanton_list]):
end:

#-----

```

```

# SEC. 3 TEST ROUTINES and CASES OF STUDY.
# EulerData(n, splitting_type, Omega_0, s_diff_0, dmax, opt_1, opt_2, check)
#-----
# REMARK. Machine used: Dell PC, Hard Drive 7 GB,
#           with Pentium II 400 MH, Cache 512 KB, and RAM 128 MB, upgrated to 384 MB on Dec. 26, 1999.
#-----

##-----
## INSTRUCTION: For a bundle V of splitting type  $[[l_1, l_2, \dots], [k_1, k_2, \dots]]$  over  $CP^n$  :
##
## (1) Omega: the Chern polynomial of the bundle V over  $CP^n$ . The indeterminate is denoted by x
##             and the hyperplane class of  $CP^n$  is denoted by h .
## (2) s_diff: the difference  $\text{rank}(U_d) - \dim(M_{\{0,0\}}(CP^n, d))$ ; see Sec. 2 of these notes for details.
##             it is a FUNCTION OF d .
##             E.g. \ 's_diff := d -> 3*d' is the function in usual notation 's_diff(d)=3d'.
## (3) x      : the indeterminate of the Chern polynomial.
##             x HAS TO BE SET TO ZERO IF s_diff IS A CONSTANT ZERO FUNCTION.
##             (Compare: Case 1 and Case 11.)
## (4) opt_1 : set to 0 if only  $K_d$  and  $n_d$  are required; set to 1 if exact  $Q_d$  is required.
## (5) opt_2 : if set to 0 then 'LSolveSub' will be employed; if set to 1 then 'LSolveSub_A' will be
##             employed. Either is a subroutine to solve a system of linear equations.
## (6) check : set to 1 if one wants to make sure that the system of linear equation is truly
##             consistent, though the theory says it must be. Otherwise set to 0.
## (7) dmax  : the maximal degree one wants to compute.
## (8) specialization: take Case 1 for example; the line ' $\lambda := i \rightarrow (i^2+7i+1)u$ ' can be
##             replaced by e.g. ' $\lambda := i \rightarrow (i^3+17i+2)u$ ' or even ' $\lambda := i \rightarrow i^3u$ '
##             as long as making sure that at least visually that none of  $\lambda(i) - \lambda(j)$  for
##              $0 \leq i < j \leq n$  is zero and that, when  $u=0$ , all  $\lambda(i)$  must be zero.
##             Usually more complicated specialization has to be used for bigger dmax value.
## (9) the first two argument of EulerData: n and splittingtype in the form  $[[\dots], [\dots]]$ .
##     Browse through the given examples below and the pattern should be clear.
##
## OUTPUT: To distinguish the two sets of numbers, the output contains built-in indications of
##          intersection numbers, "kd", and instanton numbers, "instanton_list".
##-----

## FOR NON-MAPLE-USER:
## Note that the symbol '#' de-activates all the commands or words of the same line behind this symbol.
## If you run the code as put here, it will compute only the case Case 1 since the command lines for
## all other cases are de-activated by putting a '#' at the start. (Notice the difference with Mathematica)
## To activate other cases, simply remove these '#' at the start of lines.
##
## Note, however, some of the lines are really meant to be titles or remarks and hence should be kept
## de-activated; please compare Case 1 to keep these lines de-activated.
##-----

## CAUTION: While the numerals can be changed, all the variable names have to remain as given here
##           for the compatibility with the code.
##-----

##-----
## Case 1 :  $0(5) \rightarrow CP^4$ 
##-----
restart:
print("Case 1:  $0(5) \rightarrow CP^4$ ."):
Omega  := x+5*h:
s_diff := d -> 0:
x      := 0:
opt_1  := 0:      ##### 0 or 1
opt_2  := 0:      ##### 0 or 1
check  := 1:      ##### 0 or 1
dmax   := 3:
#-----

```

```

# specialization:
# -----
if opt_1=0 then
  lambda := i -> (i^2+7*i+1)*u
fi:
#-----
qk := EulerData( 4, [[5], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
print( "Answer known n_d: 2875, 609250, 317206375, 242467530000, 22930588887625." ):
print( "Case: QK(4, [[5], []]) = [", qk[2], ", ", qk[3], "]" ):
# print(qk): ##### Re: This is de-activated for the output could be huge.
unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk',
's_diff', 'x' ):
##-----
## Date: November 25, 1999.
## Specialization: lambda := i -> (i^4+3*i+1)*u
## Answer obtained K_d (d<=3): 2875, 4876875/8, 8564575000/27.
## K_d converted to n_d: 2875, 609250, 317206375.
## Time consumed: 52,034.1 sec.
##-----

##-----
## Case 2: O(2)\oplus(4) -> CP^5
##-----
# restart:
# print("Case 2: O(2)+O(4) -> CP^5."):
# Omega := x^2+6*h*x+8*h^2:
# s_diff := d -> 0:
# x := 0:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 2:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+7*i+1)*u
# fi:
#-----
# qk := EulerData( 5, [[2,4], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Answer known n_d: 1280, 92288." ):
# print( "Case: QK(5, [[2,4], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: November 24, 1999.
## Specialization: lambda := i -> (i^2+1)*u
## Answer obtained K_d (d<=2): 1280, 92448.
## K_d converted to n_d: 1280, 92288.
## Consistency check: consistent.
## Time consumed: 6,337.1 sec.
##-----

##-----
## Case 3: O(3)\oplus O(3) -> CP^5
##-----
# restart:
# print("Case 3: O(3)\oplus O(3) -> CP^5."):
# Omega := x^2+6*h*x+9*h^2:
# s_diff := d -> 0:

```



```

# x      := 0:
# opt_1  := 0:  ##### 0 or 1
# opt_2  := 0:  ##### 0 or 1
# check  := 1:  ##### 0 or 1
# dmax   := 2:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+1)*u
# fi:
##-----
# qk := EulerData( 5, [[3,3], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Answer known n_d: 1053, 52812." ):
# print( "Case: QK(5, [[3,3],[]]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: November 24, 1999.
## Specialization: lambda := i -> (i^2+1)*u
## Answer obtained (d<=2): 1053, 423549/8.
## K_d converted to n_d: 1053, 52812.
## Consistency check: consistent.
## Time consumed: 5,693.4 sec.
##-----

##-----
## Case 4: 0(2)\oplus 0(2)\oplus 0(3) -> CP^6
##-----
# restart:
# print("Case 4: 0(2)\oplus 0(2) \oplus 0(3) -> CP^6."):
# Omega := x^3+7*h*x^2+16*h^2*x+12*h^3:
# s_diff := d -> 0:
# x      := 0:
# opt_1  := 0:  ##### 0 or 1
# opt_2  := 0:  ##### 0 or 1
# check  := 1:  ##### 0 or 1
# dmax   := 2:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+1)*u
# fi:
##-----
# qk := EulerData( 6, [[2,2,3], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Answer known n_d: 720, 22428." ):
# print( "Case: QK(6, [[2,2,3],[]]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: November 24, 1999.
## Specialization: lambda := i -> (i^2+1)*u
## Answer obtained K_d (d<=2): 720, 22518.
## K_d converted to n_d: 720, 22428.
## Consistency check: consistent.
## Time consumed: 50,205.2 sec.
##-----

##-----

```

```

## Case 5:  $0(2) \oplus 0(2) \oplus 0(2) \oplus 0(2) \rightarrow CP^7$ 
##-----
# restart:
# print("Case 5:  $0(2) \oplus 0(2) \oplus 0(2) \oplus 0(2) \rightarrow CP^7$ ."):
# Omega :=  $x^4 + 8hx^3 + 24h^2x^2 + 32h^3x + 16h^4$ :
# s_diff := d -> 0:
# x := 0:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 2:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i+1)*u
# fi:
##-----
# qk := EulerData( 7, [[2,2,2,2], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Answer known n_d: 512, 9728." ):
# print( "Case: QK(7, [[2,2,2,2],[]]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: January 8, 2000
## Specialization: lambda := i -> (i^2+13*i+1)*u
## Answer obtained K_d (d=1): 512, ---.
## K_d converted to n_d : 512.
## Consistency check: ---.
## Time consumed: 91.3 sec
## Bytes used: 22.7 MB
##-----
## Remark. The computation of K_2 exceeds the capacity of Maple V, Release 5.1.
##-----

##-----
## Case 6:  $T_{\ast}CP^2$ 
##-----
# restart:
# print("Case 6:  $T_{\ast}CP^2$ ."):
# Omega :=  $x^2 + 3hx + 3h^2$ :
# s_diff := d -> 3:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 1: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 3:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^3+17*i+1)*u
# fi:
##-----
# qk := EulerData( 2, [[1,2], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Answer known: Unknown." ):
# print( "Case: QK(2, [[1,2],[]]) = [", qk[2], ", ", qk[3], "]" ):
## print("qk = ", qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk',
#           's_diff', 'x' ):
##-----
## Date: December 8, 1999.

```

```

## Specialization:          lambda := i -> (i^2+3*i+1)*u
## Answer obtained K_d (d=1): 10*x^3
## Consistency check:      ???
## Time consumed:          ??? sec.
## Bytes consumed:         ??? MB
## Remark: For the computation of d=2, the complexity involved seems to surpass the capacity of Maple V.
##-----
## Date:                   January 7, 2000.
## Specialization:         lambda := i -> (i^3+17*i+1)*u
## Time consumed:          ???
## Bytes consumed:         ??? > 150 MB
## -----
## Remarks: So far, the job has been thrown out constantly at d=2. After pinning down exact what line
##           that cause an error message to appear and the job thrown out, it is found that, in the
##           computation for d=2, a coefficient (which is a rational function of variable 'x' and 'u')
##           that takes nearly 900 lines for the display in the Maple window appears. This seems to beat
##           the current capacity of Maple. Finding a way to go around this will be one of the major
##           improvements of the current code.
##-----

##-----
## Case 7: 0(-1)\oplus 0(-1) -> CP^1
##-----
# restart:
# print("Case 7: 0(-1)\oplus 0(-1) -> CP^1"):
# Omega := 1/(x-h)^2:
# s_diff := d -> 0:
# x      := 0:
# opt_1  := 0:      ##### 0 or 1
# opt_2  := 0:      ##### 0 or 1
# check  := 1:      ##### 0 or 1
# dmax   := 5:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i+1)*u
# fi:
##-----
# qk := EulerData( 1, [[], [-1, -1]], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Answer known: K_d = 1/(d^3)." ):
# print( "Case: QK(1, [[], [-1, -1]]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk',
#           's_diff', 'x' ):
##-----
## $K_d$ tested correct for d<=5.
##-----

##-----
## Case 8: 0(-3) -> CP^2
##-----
# restart:
# print("Case 8: 0(-3) -> CP^2"):
# Omega := 1/(x-3*h):
# s_diff := d -> 0:
# x      := 0:
# opt_1  := 0:      ##### 0 or 1
# opt_2  := 0:      ##### 0 or 1
# check  := 1:      ##### 0 or 1

```

```

# dmax := 5:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^3+i^2+3*i+1)*u
# fi:
##-----
# qk := EulerData( 2, [[], [-3]], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Answer known K_d: 3, -45/8, 244/9, -12333/64, 211878/125." ):
# print( "Case: QK(2, [[], [-3]]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: November 26, 1999.
## Specialization: lambda := i -> (i^3+i^2+3*i+1)*u
## Answer obtained K_d (d<=5): same as the known ones.
## Consistency check: consistent.
## Time consumed: 48,120 sec.
##-----

##-----
## Case 9:  $O(2) \oplus O(-2) \rightarrow CP^3$ 
##-----
# restart:
# print("Case 9:  $O(2) \oplus O(-2) \rightarrow CP^3$ ."):
# Omega := (x+2*h)/(x-2*h):
# s_diff := d -> 0:
# x := 0:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 3:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+7*i+1)*u
# fi:
##-----
# qk := EulerData( 3, [[2], [-2]], Omega, s_diff, dmax, opt_1, check ):
# print( "Answer known K_d: -4, -9/2, -328/27." ):
# print( "Case: QK(3, [[2], [-2]]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## $K_d$ tested correct for d<=3.
##-----

##-----
## Case 10:  $O(2) \oplus O(2) \oplus O(-1) \rightarrow CP^4$ 
##-----
# restart:
# print("Case 10:  $O(2) \oplus O(2) \oplus O(-1) \rightarrow CP^4$ ."):
# Omega := (x+2*h)^2/(x-h):
# s_diff := d -> 0:
# x := 0:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1

```

```

# dmax := 2:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+1)*u
# fi:
##-----
# qk := EulerData( 4, [[2,2], [-1]], Omega, s_diff, dmax, opt_1, check ):
# print( "Answer known K_d: 16, -18, 1312/27." ):
# print( "Case: QK(4, [[2,2],[-1]]) = [", qk[2], ", ", qk[3], ", ", qk[4], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## $K_d$ tested correct for d<=2
##-----

##-----
## Case 11: 0(2) -> CP^1
##-----
# restart:
# print("Case 11: 0(2) -> CP^1."):
# Omega := x+2*h:
# s_diff := d -> 3:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 2:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+1)*u
# fi:
##-----
# qk := EulerData( 1, [[2], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Case: QK(1, [[2], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: December 8, 1999.
## Specialization: lambda := i -> (i^2+1)*u
## Answer obtained K_d (d<=5): x^3, (1/8)*x^3, (1/27)*x^3, (1/64)*x^3, (1/125)*x^3
## K_d converted to n_d : x^3, 0, 0, 0, 0.
## Consistency check: consistent.
## Time consumed: 134.5 sec.
## Bytes used: 3.25 MB
##-----

##-----
## Case 12: 0(3) -> CP^2
##-----
# restart:
# print("Case 12: 0(3) -> CP^2."):
# Omega := x+3*h:
# s_diff := d -> 2:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 6:

```

```

##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+7*i+1)*u
# fi:
##-----
# qk := EulerData( 2, [[3], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Case: QK(2, [[3], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date:                ???, (Recording date: December 23, 1999).
## Specialization:      lambda := i -> (i^2+7*i+1)*u
## Answer obtained K_d (d<=6):
##   21*x^2, (189/8)*x^2, (169/9)*x^2, (1533/64)*x^2, (2646/125)*x^2, (169/8)*x^2, ---
## K_d converted to n_d: 21*x^2, 21*x^2, 18*x^2, 21*x^2, 21*x^2, 18*x^2, ---
## Consistency check:   consistent.
## Time consumed:       1,047,896.9 sec.
## Bytes used:          116 MB
##-----

##-----
## Case 13: 0(4) -> CP^3
##-----
# restart:
# print("Case 13: 0(4) -> CP^3."):
# Omega  := x+4*h:
# s_diff := 1:
# opt_1  := 0:      ##### 0 or 1
# opt_2  := 0:      ##### 0 or 1
# check  := 1:      ##### 0 or 1
# dmax   := 5:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+7*i+1)*u
# fi:
##-----
# qk := EulerData( 3, [[4], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Case: QK(3, [[4], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date:                December 8, 1999.
## Specialization:      lambda := i -> (i^2+7*i+1)*u
## Answer obtained K_d (d<=4): 320*x, 5056*x, (3893504/27)*x, 5490624*x, ---.
## K_d converted to n_d   : 320*x, 5016*x, 144192*x, 5489992*x. ---.
## Consistency check:    consistent.
## Time consumed:        55,290.4 sec
## Bytes used:           30.2 M
##-----

##-----
## Case 14: 0(6) -> CP^4
##-----
# restart:
# print("Case 14: 0(6) -> CP^4"):
# Omega  := x+6*h:

```

```

# s_diff := d -> d:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 1: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 10:
##-----
## specialization:
## -----
# if opt_1=0 then
#   lambda := i -> (i^2+7*i+1)*u
# fi:
##-----
# qk := EulerData( 4, [[6], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Case: QK(4, [[6], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: December 28, 1999.
## Specialization: lambda := i -> (i^2+7*i+1)*u
## Answer obtained K_d (d<=3): 50400*x, (752729895/4)*x^2, (433244745198080/243)*x^3, ---.
## K_d converted to n_d : 50400, 752704695/4, 433244744744480/243, ---.
## Consistency check: consistent.
## Time consumed: 49,842.4 sec
## Bytes used: 30.5 M
##-----

##-----
## Case 15: 0(7) -> CP^4
##-----
# restart:
# print("Case 15: 0(7) -> CP^4."):
# Omega := x+7*h:
# s_diff := d -> 2*d:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 10:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+11*i+1)*u
# fi:
##-----
# qk := EulerData( 4, [[7], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Case: QK(4, [[7], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: December 29, 1999.
## Specialization: lambda := i -> (i^2+11*i+1)*u
## Answer obtained K_d (d<=3): 451570*x^2, (403985396325/32)*x^4, (15755269694706695755/17496)*x^6, ---.
## K_d converted to n_d : 451570, 403983590045/32, 15755269694414078395/17496, ---
## Consistency check: consistent.
## Time consumed: 72,086.1 sec.
## Bytes used: 45.2 MB.
##-----

##-----
## Case 16: 0(8) -> CP^4

```

```

##-----
# restart:
# print("Case 16:  $\mathbb{Q}(8) \rightarrow \mathbb{CP}^4$ ."):
# Omega := x+8*h:
# s_diff := d -> 3*d:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 10:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+11*i+1)*u
# fi:
##-----
# qk := EulerData( 4, [[8], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Case: QK(4, [[8], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: December 30, 1999.
## Specialization: lambda := i -> (i^2+11*i+1)*u
## Answer obtained K_d (d<=3):
## 2773820*x^3, (3178734062035/8)*x^6, (46028387589557254161275/314928)*x^9, ---.
## K_d converted to n_d : 2773820, 3178731288215/8, 46028387589524900324795/314928, ---.
## Consistency check: consistent.
## Time consumed: 80,553.2 sec.
## Bytes used: 49.8 MB.
##-----

##-----
## Case 17:  $\mathbb{Q}(9) \rightarrow \mathbb{CP}^4$ 
##-----
# restart:
# print("Case 17:  $\mathbb{Q}(9) \rightarrow \mathbb{CP}^4$ ."):
# Omega := x+9*h:
# s_diff := d -> 4*d:
# opt_1 := 0: ##### 0 or 1
# opt_2 := 0: ##### 0 or 1
# check := 1: ##### 0 or 1
# dmax := 10:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+11*i+1)*u
# fi:
##-----
# qk := EulerData( 4, [[9], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Case: QK(4, [[9], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date: December 31, 1999.
## Specialization: lambda := i -> (i^2+11*i+1)*u
## Answer obtained K_d (d<=3):
## 13198850*x^4, (243281907041715/32)*x^8, (197802281929974511821535/17496)*x^12, ---.
## K_d converted to n_d : 13198850, 243281854246315/32, 197802281929965958966735/17496, ---.
## Consistency check: consistent.
## Time consumed: 93,665.7 sec.

```



```

## Bytes used:                    53.2 MB.
##-----

##-----
## Case 18:  $\mathbb{Q}(10) \rightarrow \mathbb{CP}^4$ 
##-----
# restart:
# print("Case 18:  $\mathbb{Q}(10) \rightarrow \mathbb{CP}^4$ ."):
# Omega := x+10*h:
# s_diff := d -> 5*d:
# opt_1 := 0:      ##### 0 or 1
# opt_2 := 0:      ##### 0 or 1
# check := 1:      ##### 0 or 1
# dmax := 10:
##-----
## specialization:
## -----
# if opt_1 = 0 then
#   lambda := i -> (i^2+11*i+1)*u
# fi:
##-----
# qk := EulerData( 4, [[10], []], Omega, s_diff, dmax, opt_1, opt_2, check ):
# print( "Case: QK(4, [[10], []]) = [", qk[2], ", ", qk[3], "]" ):
## print(qk):
# unassign( 'check', 'dmax', 'lambda', 'Omega', 'opt_1', 'opt_2', 'qk', 's_diff', 'x' ):
##-----
## Date:                    January 2, 2000.
## Specialization:          lambda := i -> (i^2+11*i+1)*u
## Answer obtained K_d (d<=3):
##   52040450*x^5, (25908993204089625/256)*x^10, (71418501571607082433686025/139968)*x^15, ---.
## K_d converted to n_d :    52040450, 25908991538795225/256, 71418501571606812655993225/139968, ---.
## Consistency check:        consistent.
## Time consumed:            80,874.0 sec.
## Bytes used:               49.6 MB.
##-----

#===== ( End of the code. )=====

```

## References

- [Au] M. Audin, *The topology of torus actions on symplectic manifolds*, Prog. Math. 93, Birkhäuser, 1991.
- [A-B] M.F. Atiyah and R. Bott, *The moment map and equivariant cohomology*, Topology, **23** (1984), pp. 1 - 28.
- [B-DC-P-P] G. Bini, C. De Concini, M. Polito, and C. Procesi, *On the work of Givental relative to mirror symmetry*, [math.AG/9805097](#).
- [C-dIO-G-P] P. Candelas, X. de la Ossa, P. Green, and L. Parkes, *A pair of Calabi-Yau manifolds as an exactly soluble superconformal field theory*, Nucl. Phys. **B359** (1991), pp. 21 - 74.
- [C-K] D.A. Cox and S. Katz, *Mirror symmetry and algebraic geometry*, Math. Surv. Mono. 68, Amer. Math. Soc. 1999.
- [C-K-Y-Z] T.-M. Chiang, A. Klemm, S.-T. Yau, and E. Zaslow, *Local mirror symmetry: calculations and interpretations*, [hep-th/9903053](#).
- [F-P] W. Fulton and R. Pandharipande, *Notes on stable maps and quantum cohomology*, in *Algebraic geometry - Santa Cruz 1995*, J. Kollár, R. Lazarsfeld, and D. Morrison eds., Proc. Symp. Pure Math. vol. 62, part 2, pp. 45 - 96, Amer. Math. Soc. 1997.
- [Gi] A. Givental, *Equivariant Gromov-Witten invariants*, [alg-geom/9603021](#).
- [H-K-T-Y] S. Hosono, A. Klemm, S. Theisen, and S.-T. Yau, *Mirror symmetry, mirror map and applications to complete intersection Calabi-Yau spaces*, pp. 545 - 606 in *Mirror symmetry II*, B. Greene and S.T. Yau eds., Amer. Math. Soc. and International Press, 1997.
- [Ko] M. Kontsevich, *Enumeration of rational curves via torus actions*, in *The moduli space of curves*, R. Dijkgraaf, C. Faber, and G. van der Geer eds., pp. 335 - 368, Birkhäuser, 1995.
- [Li] Kefeng Liu, private communications.
- [L-L-Y] B.H. Lian, K. Liu, and S.-T. Yau, *Mirror principle I*, Asian J. Math. **1** (1997), pp. 729 - 763; *II*, [math.AG/9905006](#); *III*, [math.AG/9912038](#).
- [M-P] D.R. Morrison and M. Plesser, *Summing the instantons: quantum cohomology and mirror symmetry in toric varieties*, Nucl. Phys. **B440** (1995), pp. 279 - 354.
- [MS] *Mirror symmetry I*, S.-T. Yau ed., Amer. Math. Soc. and International Press, 1998; *Mirror symmetry II*, B. Greene and S.T. Yau eds., Amer. Math. Soc. and International Press, 1997; *Mirror symmetry III*, D.H. Phong, L.V. Vinet, and S.-T. Yau eds., Amer. Math. Soc., International Press, and Centre de Recherches Math., 1999
- [Pa] R. Pandharipande, *Rational curves on hypersurfaces (after givental)*, [math.AG/9806133](#).
- [Wi] E. Witten, *Phases of  $N = 2$  theories in two dimensions*, Nucl. Phys. **B403** (1993), pp. 159 - 222.

—————( Maple V )—————

- [H-H-R] K.M. Heal, M.L. Hansen, and K.M. Rickard, *Maple V - Learning guide*, Waterloo Maple Inc., and Springer-Verlag, 1998.
- [M-G-H-L-V] M.B. Monagan, K.O. Geddes, K.M. Heal, G. Labahn, and S.M. Vorkoetter, *Maple V - programming guide*, Waterloo Maple Inc., and Springer-Verlag, 1998.